User's Guide

3D Game Creation System

for DOS

pie in the sky software

Developers of 3D Entertainment Software

TABLE OF CONTENTS:

CREDITS 5

CUSTOMER SERVICE 5

QUICK START 6
- Warnings 6
- Installation 6
- Configuration Message 6
- Running the GCS 6
- Crashes, Hangs, & Errors 7
- The 3D World 8
- The ICONS 8
  - The select and view icons 9
  - The object placing icons 9
  - The magnify icons 10
  - The fading table icon 10
  - The editing icons 10
  - The floor/ceiling icons 10
  - The grid icons 11

NAMING AND MAKING YOUR PROJECT 11
- Putting walls and objects in your new level 13
- Putting walls into your layout 13
- Move around in Zoom mode 14
- Putting up trees and other stand-up items 14
- Solid color horizontal panels 14
- Platforms: steps and teleporters 15
- Animated objects 15
- Enemy characters 15

LAYING OUT A GOOD 3D GAME LEVEL 16
- Designing the levels 16
- Outside Scenarios 17
- Backg??.vga for your sky or ground 17
- Buildings 18

EDITING YOUR 3D LEVELS 19
- Moving/copying walls around 19
- Grid snap 19
- Object snap 19
- Selecting objects 19
- Moving walls or whole rooms 20
- How to copy groups of objects 21
- How to erase groups of objects 21
- Rotating groups of objects about a point 21
- Altitude Adjustment 21
- Changing a placed wall into a door 22
- The Find All function 22
- Invert 22
- Editing object attributes 22
  - Don't Draw Backsides 22
  - Don't Fade With Distance 23
  - Don't Draw If Far Away 23
  - Can't Bump Into Object 23
  - Set Wall Lighting By Angle 23
  - Object Won't Show Up On Radar 23
  - Set FTC On Solid Walls 23

ADDING DOORS TO YOUR GAME 24
- How to Make a Door 24
  - Resistance to forced entry 24
  - Number of Seconds to Stay Open 25
  - Key Value 25
  - Open upon Explosion 25
- Tips on placing doors 25
  - Door Frames 25
- How to avoid graphical problems with doors 26

UNIVERSE REGISTERS 26
- Special purpose registers 27

OBJECT LIBRARIES 28
What is an object library? 28
Choosing an object library 28
Importing or creating new artwork 28
Modifying an object in an object library 29
Removing an object from an object library. 30
Adding Animated objects to an object library 30

TEXT MESSAGES FOR THE PLAYER 30
Editing the pstr.txt file 30
Compiling your messages with pstr.exe 31
Getting more specific about pstr.exe 32

PUTTING ENEMY CHARACTERS IN YOUR GAME 33
Quickness 33
Step size 33
Resistance 33
Attack Strength 33
Attack Accuracy 33
Range 33
Enemy Character Behavior 33
Sentry, then hunt 34
Random Patrol 34
Stand and shoot 34
Enemy Artificial Intelligence 34
What alerts an enemy? 34
Enemy character sounds 35
How enemies behave 35
Making your own enemies 35
How to modify the appearance of the enemies 36

WEAPONS 37
How weapons work 38
Customizing the weapons 38

MAKING ANIMATED OBJECTS 40
How to make an animated object 41
Summary of Animation Commands 42
Unconditional Branch 42
Branch if Shot 42
Invisible Blast 43
Remove this Animation 43
Add a value to a Register 43
Branch if the Player Close 43
Warp to a New Level 43
Damage Player 43
Play Sound Number 43
Skip Next if Reg != Val 44
Branch if no Guards 44
Skip if Player Close 44
Set Invincibility 44
Assign a value to a reg 44
Skip Next if reg=val 44
Display a Text Message 44
Modify a lighting register 44
An example of a typical animated object 45
More info about Animated Objects 46
How to use your animated object 47
What to do if your object causes Critical Errors 47

MAKING SOUND WITH PIE 3D GCS 48
Importing your .WAV sound files 48
Making Music with the Pie 3D GCS 49
Problems with midi music files 49

INVENTORY ITEMS 50
Complete list of the default placeable objects 51
Goo Gun 51
Shotgun 51
Machine Gun 51

Hand Grenade 51
Rocket Launcher 51
Generic Ammo 51
Shotgun & Machine Gun Ammo 51
Rocket Ammo 51
Gas Grenades 51
Bomb 51
First Aid 51
Key 51
Document 51
Armor 51
Gas Mask 51
Radar Pack 51
Customizing appearance of inventory items 52
Customizing the function of Inventory Items 52
Making your own special inventory items 54

ARTWORK FORMATS AND ISSUES 59
What is an image file? 59
What is resolution? 60
Why is resolution so important? 60
Memory Limitations and Image Resolution 61
Getting image files into the 3D world 62
Images vs. Objects 62
Bitmap image resolution vs. 3D world size 62
Typical resolution and world sizes 62
Walls with shapes and holes 63
Limitations in the shapes 63
Shapes & holes to avoid 64
Symmetrical Objects 64
Ceiling and floor images 64
VGA vs. VGR image files 65

GCSPAINT 66
GCSPAINT help 66
Disk commands 66
Create New Image 66
Save Image 66
Merge Image 67
View as Tile 67
Hidden Main Menu 67
Palette 67
Select a Color 68
Hidden Palette Box 68
Load a Palette 68
Modify Palette 68
Match Palettes 68
Save Palette 68
Draw Tools 68
Interpolation 69
Settings 69
Text 69
Color tools 69
Fill 69
Color Replacement 69
Random Replacement 69
Color Phase 69
Color Sunburst 69
Special effects 69
Outline 70
Anti-Aliasing 70
Merge/Blend 70
Brighten/Darken 70
Smooth Image 70
Contrast 70
Rectangle tools 70
Zoom 71
Oops 71

LIGHTING EFFECTS 71
Lighting registers 71
How to do lighting effects 72

MAKING THE FINAL GAME 74
Connecting the levels 74

Warp-to Points 74
Important considerations 75
No Level switches in test mode. 75
Make Final will NOT work unless you have specified a game start position. 76
Make Final will NOT work unless you have tested all your levels. . . . 76
Setting the Game Start position 76
The correct sequence of preparing for Make Final 77
Trying out your final game 77
Go.bat command line options 77
What to do if there are problems 77
Common problems in the final game 78
Legal notice of what you may distribute 78
PKZIP 79
Putting your game on floppies 79

CUSTOMIZING OTHER PARTS OF YOUR GAME . . . 80
Making shading effects: fog & dark 80
Using a custom palette 81
Using a Custom background textures 81
Customizing the hit point guy 82
Customizing the game play screen 82
Customizing the Help and Pause screens 83
Customizing the number readouts 83
Menu for the final game and SAVE/LOAD 83

NOT ENOUGH MEMORY? 83
Freeing Up Available Memory with CONFIG.SYS 84
Info about Available XMS or EMS ram. 85
Why PC memory is so weird 86

TOP TEN TROUBLE SHOOTING SOLUTIONS 88

CRITICAL ERRORS 90
Most common errors 92
Raw Critical Error list 94

INDEX 97

CREDITS

GCS Programming.....John Nagle & Kevin Stokes

3D Engine Programming....Kevin Stokes & Mat McKenzie

GCSPAINT.....David Johndrow

Artwork.....Colin Stokes & Terri Hamel

Music.....John Davis

Portions of this product were developed using Fastgraph Programmer's Graphics Library. We heartily recommend this excellent product from Ted Gruber Software. Call 1-800-410--0192 for more information about Fastgraph.

CUSTOMER SERVICE

Thank you for giving us your business. We value you as a customer and we will go the extra mile to make sure that you remain satisfied with your purchase.

Here are our customer service phone numbers...

Call our ORDER LINE 1-800-537-3344 to order additional Pie in the Sky software.
Call our SHIPPING DEPT. 716-425-8510 for name and address corrections, replacement disks, and billing & shipping
Call our TECH SUPPORT LINE 716-425-8782 if you have questions about using the GCS.
Your first three calls are free. Make sure you have your serial number ready. For billed calls you will need your MC/VISA number and expiration date.
Call our BBS at 716-425-2962 for new demos, product announcements, and other services.

You can also send us e-mail via our PRODIGY address DWHC72A.

QUICK START

The first step in using the GCS is installing the program on your hard drive. The GCS takes under 4 megabytes of hard disk space, although this will grow as your games grow.

Warnings

Do not install the GCS under Windows. This is a DOS product. Exit Windows before installing the GCS.

Make sure you have at least 4 megabytes of space left on your hard drive before installing.

Installation

To install the Pie 3D Game Creation System, just put "PIE 3D GCS - Install disk 1" into your floppy drive, and type

A:install

at your DOS prompt. Follow the instructions on the screen to get the program installed on your hard drive.

Configuration Message

The installation checks your computer's configuration to make sure that the GCS will run properly. If your configuration does not meet the GCS's requirements, a menu screen appears. If this menu screen does not appear, your configuration is adequate and you can go ahead and run the GCS.

The configuration menu screen gives you four options.

If you choose option #1, the program helps you make a boot disk with a configuration that will allow you to run the GCS. You will need a blank floppy disk. Follow the instructions on the screen to make your GCS boot disk. Whenever you want to run the GCS, start up your computer with this boot disk. The advantage to making a separate boot disk is that you don't run the risk of messing up your original config.sys and autoexec.bat files on your hard drive.

If you choose option #2, you can run the GCS under your current configuration. The worst thing that can happen is that the GCS will display a critical error message when you start up the 3D game engine when it comes time to test a level. You may get lucky and the GCS will run fine. You can always run the program HELP_ME.EXE from the main P3DGCS directory to get tips on how fix configuration problems later.

If you choose option #3, you will go back to your DOS prompt. If you want to read more information about configuration

issues before proceding, please run HELP_ME.EXE from the main P3DGCS directory.

If you choose option #4, you will see another menu. You can now choose to read detailed information about specific configuration issues such as mouse drivers, sound cards, and memory.

You should also read the section called "NOT ENOUGH MEMORY?" later in this manual. In addition, please consult the section called "TOP TEN TROUBLE SHOOTING SOLUTIONS" near the end of the manual.

We want you to be a happy GCS user. That is why we have provided the boot disk option and this helpful documentation. Be patient. Read carefully. Don't get discouraged. If you follow the procedures described above, you should be able to get the GCS up and running on your own, despite your computer's incompatible configuration.

Running the GCS

Now you should be all ready to go. Type GCS to bring up the Game Creation System for the first time. Click the 'OK' box to get started.

!!! NOTE !!! If the OK button does not work, see the chapter entitled 'Top Ten Trouble Shooting Solutions' near the end of this manual.

The first thing you encounter is a red box which asks you to specify a project. Right now, we want to see the demo, so click on the 'show list' button, and then click on 'DEMO'. That just types the word 'demo' for you. Now just click on the 'accept' button to go ahead and open the DEMO project.

Now a red box comes up warning you that the colors are going to change on your screen. This is normal as the GCS adjusts itself to use the same colors the DEMO game will use. Click on the OK button of this red box.

Now you should be looking at the main GCS screen. There are pull down menus across the top of the screen. Icon buttons are on the left side of the screen. And, a big black empty viewport is in the middle. There is one white layered square in the center of the screen. You are looking at an empty level named 'UNTITLED'. Just like a word processor program usually opens a document called untitled when you call it up, GCS brings up an empty level. To actually load the level we want to look at, click on the FILE menu in the upper left part of the screen. After you have pulled down the file menu by clicking on the word File, click on the 'open level' command.

Right away you get a red box asking if you want to save the current level. Well, there is no sense in saving an empty level, so reply no by clicking on the no button. In general when you start the GCS, you will be clicking on 'open level.' Unless you have added objects to the untitled level, always say 'no' to this red text box at this point.

Now you are back to the box that says 'Choose Project'. The word DEMO should already be typed for you, so just click on the 'ACCEPT' button. Next you will be asked to select a level from a scrolling list. However, this demo has only one level. Click on 'LEVEL1.WLD', then click on the 'ACCEPT' button.

Boom! You should now be faced with a top view of a level. The yellow lines are walls drawn from the top view like a blueprint of a house. The blue squares are inventory items that the player can pick up. The little yellow squares are enemy guards.

If you want to see what each wall actually is, click on the icon on the top row that looks like a question mark and a little box. Then move the mouse pointer around on the level. As the little white line jumps around from wall section to wall section, a picture of that wall appears in the lower left of the screen.

Now press the right mouse button to get out of 'view' mode. Notice that when you are in view mode, the question mark icon stays depressed. This is telling you that you are in view mode. When you clicked on the right mouse button, you left view mode and entered selection mode. The big arrow icon is now depressed to show you that you are now in selection mode.

You can use the arrow keys to scroll the top view around. The keypad + and - keys zoom in and out.

Before you modify the level, it would be fun to jump into the 3D world to see this level in 3D. Pull down the 'FILE' menu, and press 'test level'. A red box comes up asking if you want Sound. If you have a 100% Sound Blaster compatible sound card, you may want to choose yes. Then another box comes up that asks if you want to test the game in God mode. God mode is a term used in other games for a long time to mean that the player is impervious to damage. This can be useful for testing purposes. Click on either yes or no.

Now a bunch of text will come up on the screen. This is the 3D engine starting up. It will be busy for a few seconds, as it loads up the artwork necessary to bring up the game.

Crashes, Hangs, & Errors

If you get a Critical Error #92 at this point, then probably you need a line 'FILES=30' in your CONFIG.SYS file in the root directory of your hard drive. Read your DOS manual about this command. You will need to reboot your computer after making this change. Before modifying your config.sys file, make sure you have a backup boot disk containing your original config.sys and autoexec.bat files. Of course, if you choose option #1 from the configuration menu described above, you won't have to deal with this problem. If it fails with other critical errors, or just goes back to the GCS program, then try again with sound off. If you had to bypass the memory check to start the GCS earlier, it is possible you just need to free up some more conventional DOS ram. If so, read the chapter on memory issues. Look up critical errors in this manual.

The 3D World

Finally, the screen should go black and come back in the 3D world. Right away, you can see the floor and ceiling textures. Go out the door and into the red brick hallway. You will see a red key on the floor of the hallway. You pick it up by getting close to it. The red key icon shows up on the left side of your screen. When you see it there, press the 'I' key. This will bring the key into your inventory so you will have it when you get to the door that needs it.

If you want to put the red key down, press 's'. Then, use the arrow keys to move the underline under the key icon. Then press return to drop the key in the 3D universe.

Go straight across the hall without turning right or left, and pick up some weapons and ammo. Pick up the weapons first, so the ammo will be automatically picked up and loaded into your weapons. If you pick up the ammo first, then you will have to drop the ammo and pick it up again to reload.

Now go back and turn right and go through the grey door. Here, you will find the red key card that you will need to get into the main area where the enemies are. You will see a bunch of health packs, and a red key card. The health packs will repair damage sustained by enemy gunfire. When you are wounded, you have to drop them and then pick them up again to restore your health.

Now with the red key card, you can go through the metal door and confront the bad guys. The shotgun should make short work of most of them if used at short range. Use the machine gun if you don't want to get in close. Fire the weapon by pressing the space bar. You switch weapons with the number keys across the top of your keyboard. These are NOT function keys, or keypad keys. Press '1' for your karate kick, 2 for the goo gun, 4 for the machine gun, and 7 for the shotgun.

If you hit escape, or get killed, you will be whisked back to the GCS, Your level is just the way you left it.

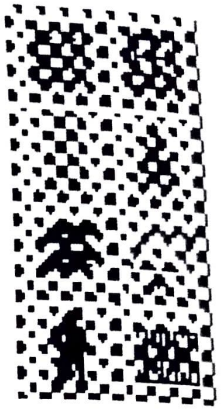
If you want to start playing with the icons and changing the level around, it might be a good idea to make a copy first. Click on the SAVE option in the file menu. But instead of just ACCEPTING the name level, you can save it as another name. Then feel free to modify it without the worry of losing the original.

Experimenting is a fun way to learn with the GCS. Take some time to get to learn how the GCS operates before settling down on your big project. Try clicking on the brick wall icon to place some new walls, or the tree icon to place some plants or other objects. In general, the right mouse click should get you out of most things.

The ICONS The select and view icons

The select icon is the arrow button. When you select this, you are ready to select objects from the viewport to move, copy, elevate, rotate etc. This is the 'top level' of the GCS. This means that upon completion or cancellation of most commands, you automatically return to this 'select' mode. For more information about select mode, see the chapter on editing. The icon with the question mark is the view icon. When you click on this icon, you go into view mode. Then as you move around on the viewport, you can see what the walls look like by watching the lower left

corner of the screen.



The object placing icons

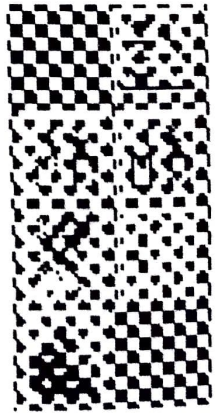
These icons are what you click on when you want to add a new object to your 3D level. Click on the brick wall icons to add walls to your 3D levels. The plain brick wall is for walls that are solid from floor to ceiling. The windowed wall icon is for walls that have black pixels (COLOR #0) which are invisible in the 3D world. If you make a wall with a black rectangle in the middle, the wall will have a window cut out that the player can see through. The door icon is not really a placing icon, since it operates on walls you have already placed. See the editing icons for this one. The tree icon is for placing stand-up objects like trees or lamps in your level. The flat squares are for solid color floor/ceiling polygons. The wire frame square icon is for placing platforms. The guy icon is for placing enemies, and the film is for placing animated objects that you have already created and put into your library.

The magnify icons

The magnify icons are for zooming in and out on your top view layout. The + magnifying glass zooms in. The - magnifying glass zooms out. When you click on one of these icons, you enter 'magnify mode'. Each left-click zooms you in (or out) again. The location of your mouse pointer, when you click the button, is where the new screen center will be.

The fading table icon

Click on this icon to set the fading color for this level. When you click on this, you will be asked to choose a color that your level will fade to. IE, in most 3D games, walls darken with distance until totally black when far away. The Pie 3D GCS engine will let you to fade to any color with distance. The most useful colors are black, grey and white. The fade to grey gives you the fog/rain effect in outside scenarios. Fading to white is nice for a smokey air effect, or a sterile environment. You must do this at least once with each of your game levels. Usually when you change your fade color, you will want to modify your BACKG?.VGA file. See the chapter on customization for more information about that.



The editing icons

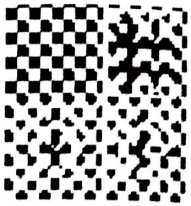
These icons all operate on items that you have selected while in select mode. The top icon is the set elevation icon. This is for moving your walls up and down. The next icon is for moving groups of objects. The next icon is for copying objects. It works the same as the move icon, except that it leaves the original objects where they were, and makes a copy somewhere else. The eraser icon is for removing objects, and the circular arrow icon is for rotating groups of objects by 90 degree increments. The bag icon is not really an

editing icon. It belongs with the placing icons. It is used for placing inventory items. As for why it isn't in with the placing icons, we defer to Ralph Waldo Emerson: " A foolish consistency is the hobgoblin of little minds." See the chapter on editing for more information about the editing icons. See the chapter on inventory items for more info about the bag icon.

The floor/ceiling icons

These icons create a repeating floor or ceiling tile. The original artwork used must have a resolution of 64 pixels wide by 64 pixels tall. This is unrelated to the 3D world size in cm. It is the size of the image in GCSPAINT that must be 64x64 pixels. Texture mapped ceilings can only be used in levels that have walls that are 400 cm high. Levels with taller walls will have to use the fixed flat image ceiling in the file backg??.vga as described in the Custom chapter in this manual.

The grid icons



Later in the manual, the grid will be covered in more detail. The icon that looks like a tic-tac-toe board sets the grid spacing, usually 100, 200 or 400. This setting doesn't affect wall placing grids, but it will affect grids for placing trees and other stand-up objects. It will also will set the grid for the editing commands. The 'O' and 'G' icons toggle the snap to object and snap to grid features. For further info on the grid, see the chapter on editing.

NAMING AND MAKING YOUR PROJECT

With the Pie 3D GCS, you can make as many games as you like. Each game is kept in a separate directory. These are called 'project' directories. Each project is an entire game. Each project can have multiple levels. For example, you might call your project 'SNOW', and have a game with three wintertime levels in it. You might also have a project called 'CAVEMAN' with five levels of cave mazes, and caveman objects.

So to start making your game, you must first tell the GCS the name of your project. A project directory will be created for you. If you would like to start a new project, start from the DOS command line, and run the GCS. The first dialog box that comes up asks you to select a project. If you want to make a new one, just click on the name box, and make up a name for your new project that is eight or fewer characters. Then either hit the enter key, or click on the 'ACCEPT' button.

When you hit the 'ACCEPT' button, the GCS then creates a directory on your hard drive filled with files that the 3D engine needs to enter the 3D world. This directory will have the same name as your project name. So if you made a project called myproj, then a new directory would be created. Assuming you installed the GCS in the directory c:\p3dgcs, this would be the full name of the project directory.

c:\p3dgcs\engine\myproj

Many files are put in this project directory. These include the picture files for your game's background, the damage indicator artwork, and so on. At the beginning, these files are their stock entities. Once you have created your directory, you may edit them as you like if you want a custom background or damage indicator etc.

Next the GCS asks you to pick a palette for your new game. A palette is a set of colors. $rp9a.pal is the general purpose palette. For most applications, probably $rp9a will suit your needs. If you want to know about what a palette is, read on.

Each set of artwork is kept in a separate directory on your hard drive. All the object libraries and enemy directories are specific to certain palettes, or color sets. If you tried to mix images from different palettes, then the colors simply come out all wrong, and the enemies, walls, or whatever will come out looking like fluorescent soup.

This is not a limitation imposed by the Pie 3D game engine. The VGA 256 color mode can only put 256 unique colors on the screen simultaneously. Different sets of artwork might use different sets of 256 colors. For example, for a set of artwork made for wintertime games you might want lots of shades of white and blue. That way all your ice and snow artwork can have lots of subtle variation in colors. For your dungeon game, you probably want your 256 color palette to have lots of browns, and mossy greens. There wouldn't be a lot of use for the blues and bright whites.

To keep artwork together that all uses the same palette, there is a separate master directory for each palette. The enemy directories, and object libraries are all kept in directories that are inside the master palette directory on your hard drive. Once you choose a palette for a project, there is no way to change it. Since you cannot change a palette without rematching every piece of artwork in your game, it would always be easier to start a new project.

Later on, when you are more familiar with the GCS, you may want to create your own palette directory, and use your own set of 256 colors for your game, instead of choosing one of ours. Please do not attempt this unless you have already become proficient at using the GCS. If you are using the GCS for the first time, just skip ahead to the next paragraph. To make a new palette directory, first use GCSPAINT to make and save the palette file with any filename. Then you exit to DOS, and use the DOS command MKDIR to make a subdirectory of the same name as your palette. This subdirectory should be off the main P3DGCS directory. Now you have made your new palette directory, you must start a new project to use it. Your new palette directory name should appear when you click on the 'show list' button when choosing a palette. Since your palette file has nothing in it, the first thing you must do is create a new weapon set, and new guard set, and a new library directory. All of these need to be in your palette directory in order for you to make a game that uses your new palette. Note also, that your new palette directory will have no artwork in it at all, so you will have to either copy some .VGR files into your newly created library file, and import them, or palette match some from the other palette directories, and import those.

Once you have selected which palette you want to use for your project, you are asked to choose an object library. Unlike the palette choice, you can switch libraries anytime. Object libraries are just what they sound like. They are directories filled with

artwork images. Object libraries also have a system for storing the sizes of an objects.

Choose any library from the list. You can browse through the libraries in your palette directory easily, adding a wall from one library, and then switching to another. After selecting the library, you are left facing a blank world from the top view. The little box thing in the middle of the screen is the default player starting position, in the exact middle of the level.

Nature abhors a vacuum. In the same way, the GCS abhors a project with no levels. Therefore, when it creates a new project, it will create an empty level named 'untitled'. You can start editing right away. When you think you've created something you want to keep, use the 'save' button to save the level with a real name. You cannot save your level with the name 'untitled', so make sure you give your level a new name when it is time to save it.

Putting walls and objects in your new level

All the trees, enemies, and wall pieces that make up the 3D world are called 'objects'. Rooms and hallways must be made from fixed-length sections of textured wall. It is easy to predefine the sizes of textured walls or other objects, but that will be discussed later. The point is that doing your 3D layout is simply a matter of selecting the object you want to place, and then placing it.

Putting walls into your layout

Now you are ready to start laying out your first level. To get started right away, just click on the solid brick wall icon. Then select a wall texture from the scrolling list. As you move the mouse over the items on the list (without clicking), pictures of walls will appear to the right of the scroll box. When you find one you like, click on the name in the scrolling list. When you click on the name, you are then in 'place' mode. The menu bar at the top of the screen goes away, and a text message appears telling you to choose an anchor point for your wall. In addition, a grid will appear in the viewport to help you align your walls. As you move your mouse around on the viewport, you will be dragging your wall around. Unlike other programs, you do not hold the mouse button down to drag things. As you drag the wall around, it will always hang in an east-facing orientation. You first must left-click to fasten the lower end of the wall to the grid. Now you will see another wall section is now there. It is following the mouse. It is attached to the first wall you put down. Now you have three options. You can...

1. get out of wall-placing mode with two right-clicks. This leaves your first wall, but cancels that second one that appeared. Or, you can...

2. detach the new wall section with one right-click. Then you are dragging it around, ready to anchor the bottom end. Or, you can...

3. choose a direction for the new wall, and left click to make it permanent. After doing this, you have placed two walls that are attached.

When you are starting a new layout, you will want to make big rooms and hallways with all the same wall type. Thus, you would choose option three above. You can always go back later, delete a wall here and there, and replace it with another type for variation.

Move around in Zoom mode

A thing to remember about place mode, is that you can scroll up, down, or side-to-side by clicking on either of the magnifying glass icons. If you want to move your view of your level, click on one of the magnifying glass icons. The cursor will turn into a cross. Clicking the left mouse button will zoom you in or out depending on which magnifying glass icon you selected. The '+' one zooms you in, and the '-' one zooms you out. When you are in magnify mode, the magnify buttons remain down. You can also use the cursor keys or the keypad '+' and '-' keys to zoom in and out. You can leave magnify mode by right clicking on the mouse. Then you will put back into place mode.

Putting up trees and other stand-up items

The other icon is a brick wall with a window on it. This is used for walls that have holes in them, such as a wall with a window or a fence with a rough top. Placing these objects in your level is the same as placing the solid walls.

The third tool icon in the GCS is for drawing stand-up objects like trees or floor lamps etc. These are shaped objects that look the same from every direction. When you click on the icon, you are presented with a list of object names. Moving the mouse pointer across the list will show the images in the lower left-hand corner of the screen. Choosing one item puts you in place mode.

This place mode is a little different though. The direction doesn't matter at for these stand-up items since they always rotate to face you. Therefore, there is no reason to anchor one end first, and then swing it to the proper direction. To place the stand-up objects, drag the square to the place you want, and left-click to place it. The square shows a rough representation of the horizontal width of the stand-up object. This is so you can judge its distance from walls. Once you left-click to place the stand-up, another potential stand-up square appears attached to your mouse pointer. You can drag and place this one, or you can right click to get out of place mode.

Note, you usually want to turn off the grid before you decide to click on the tree icon. When placing things like floor lamps, wastebaskets and other things, the grid is usually more of a hindrance than a help. You can click on the grid icon to toggle the snap to the grid feature on and off.

Solid color horizontal panels

The PIE 3D GCS can have texture-mapped floors and ceilings. However, you can only have one repeating pattern on the floor and ceiling. For the situations where you really want to have a section of the floor look different from the rest of the floor, you

can use the solid color polygons. A good example might be an elevator. You might have a nice carpet-like textured image spread over the floor, but you don't want the carpeting to extend into your elevator! The solution is to place a solid color floor polygon on the floor of the elevator.

Click on the tool icon that looks like two purple sheets suspended in the air. Choose a color from the color box that pops up. Click twice to specify the corners of the rectangular floor areas that you want covered. Of all the objects that the 3D engine draws, the floor polygons are the most finicky. Polygons that cover huge areas of the floor can be unstable at certain viewing positions and angles. Often they will work better when broken into smaller overlapping pieces.

Platforms: steps and teleporters

Platforms are rectangular areas in the 3D world that sense the presence of the player. One of the most common uses of a platform is to raise the player when he steps upon one. Another function is to wait until a player crosses a rectangular area triggering a teleport to another level. In fact, this is how most level-switches are implemented with the GCS.

The rectangular platforms are, by default, not drawn; they are completely invisible. Usually, they are placed right in front of tunnels or staircases. When the player walks up to the staircase, the game switches levels as if he or she had traversed the staircase.

As mentioned above, another common use is to raise the player. Let's say you create a box out of wall panels, and you want the player to be able to jump up on the box and stand there. Without a platform, the player would sink right down into the box instead of standing on top of it. The box needs to have a platform over it so that when the player is within the platform's boundary, his feet are lifted to the height of the platform's z value.

Still, another use for platforms is to set universe registers. These are values that animated objects use to communicate events. For example, if you wanted to trigger an explosion when the player walks through a particular are. Just make a platform that sets univ register 5 to value 1. Then make an animated object that tests univ register 5, and when it gets to value 1, it blows up. Of course you don't have to use this feature at all if you don't want to. Universe registers will be covered in detail elsewhere in the manual.

To create and place a platform object, you click on the tool icon that looks like a suspended square wire frame. A dialog box comes up with some options. You can set the platform to be visible. This actually just makes a floor polygon in the same place as the platform. If you choose to do this, you will be asked to pick a color. In addition, you will have to choose the function of the platform. Warping to a different level is a common one. If you select this, then you will have to know the entry point number of the destination level. Don't worry if you don't know what an entry point is at this point. They become important only when you are trying to connect levels with staircases and other passages. This will be covered elsewhere in the manual.

Animated objects Placing animated objects is easy, but it requires a whole chapter of explanation.

Enemy characters

Placing enemy characters is almost as easy as placing stand-up objects. You get a dialog box of options for each enemy. When you finish and click on the OK button, you are put into place mode. You are asked to chose a direction for the enemy to face. North is towards the top of the screen. After you decide on the facing direction, you click where you want your enemy to be in your level. Most of the time, you will want to turn the grid off before placing enemies. Otherwise, the layout editor may place your enemies in the middle of walls when attempting to align your enemy with the grid.

After you place them, the enemies can be copied, moved and rotated like any other object using the normal copy, move and rotate icons. However, there is a maximum number of 32 enemies on any one level. The enemy characters in the Pie 3D GCS can patrol about randomly, or stay sentried in one spot. If they notice you, they will start shooting and chasing you. As game designer, you can control their behavior by setting their parameters. You can also change their appearance by editing their artwork files.

For more information, see the chapter that is completely dedicated to enemies.

LAYING OUT A GOOD 3D GAME LEVEL

With the Pie 3D GCS, it is very easy to make great looking levels very quickly. This chapter covers the basics of how to go about making various kinds of levels. Making a maze level with equal sized walls is as simple as clicking on the solid wall icon, and clicking away until the maze is done.

The first game level should have a starting place for the player. Then, somewhere in the maze, there should be a passage to the next level. You can actually have as many staircases or elevators as you like, and they can go to any level, not just the next one. But to make a basic game, the first level should have one staircase going to the next level. Each level after that should have one staircase going back up to the previous level, and a staircase going down to the next level. At the last level, there must be a trigger to end the game.

Designing the levels

This part is up to you. To start with, we recommend that you pick out several wall panels of the same size, 400x400 cm. When all the wall panels are the same size, you can make your levels very quickly, and the grid snap will make sure that your walls will close beautifully.

Each level has a maximum number of 700 total objects. If you want to fill the entire level space with a dense maze, you find that you run out of object space before you are finished. If that happens to you, it is helpful to make a double wall panel that is

800 cm wide instead of 400 cm. Then you can replace two single walls with one double-wide wall. This reduces the total number of objects in the level. The tradeoff is that the double wall panels take up more RAM, because they usually have more pixels in the artwork files. It is important to distinguish between the two limitations. The basic RAM limitation is that all your artwork must fit into DOS memory. The more pixels in your artwork, the quicker your RAM will be used up. The 'M' bar meter on the bottom of the GCS screen shows how much RAM you have left. Each time you add a new wall, you will see your 'M' meter increase. When the meter increases to the end, you cannot add any more different objects to your game. The 'M' meter is based on your system's available DOS ram. Therefore, if you run the GCS with different amounts of RAM at startup time, you will get different results with your 'M' meter. Also, testing your world with sound on uses significantly more RAM, so if you test with sound on, then you will see a higher reading on your 'M' meter. Due to technical issues, the 'M' meter is most accurate right after you come back from a test. If you have been adding and removing many objects, it can get inaccurate, thus requiring another 'test ' command.

If the GCS starts beeping at you, it is trying to warn you that you may have loaded too many unique pieces of artwork. The 3D engine won't work properly at test time or in 'MAKE FINAL'. The solution is to remove all instances of an object to decrease RAM requirements. Be careful with animated objects, too, since the GCS must load all those pieces of artwork.

The 'W' meter measures the number of objects you have placed in your 3D world. This is very different from the 'M' meter, because the 'W' meter doesn't care whether you are placing a wall you have used before on that level. Any object placed into 3D world counts as one. A tiny key on the floor counts as one, the same as a huge wall panel. Adding an enemy or a platform or a solid color floor panel also counts as one. You cannot have more than 700 total objects on a level. When your 'W' meter maxs out, you cannot add any more objects to your 3D world without removing something. That is why it is helpful to use a wall that is 800 cm wide for large rooms and long hallways.

Although steps with platforms are possible, we recommend that all users initially stick with the basic maze layouts until they become familiar with the GCS.

Outside Scenarios

Outside scenarios can be very easy and fun to build. The simplest scenario is one where there is nothing but trees. Just make a tree image, save it, then import it into your object library. Then click on the tree icon to start placing the trees all over your level.

You can hit 'test level' to walk around your forest. If you haven't set a floor, you will get a shimmering grainy pattern for a ground cover, and the sky will be a solid color. These are not regular texture-mapped graphics. The solid color sky and the grainy floor are part of a .vga file called 'BACKG??.VGA' in your project directory. The '??' convention refers to your level number.

Backg??.vga for your sky or ground.

Once you have created your project directory, you are free to edit that backg??.vga file to your own tastes.

If you looking at the various BACKG??.VGA files with GCSPAINT, you will see different colors used for different purposes. For indoor levels where both the floor and ceilings are specified as repeating panels, the backg??.vga files don't do much. However, wherever you have a high ceiling, no ceiling at all, or no specified floor or ceiling, the pattern in backg??.vga will be highly visible.

If you don't specify any floor or ceiling, then backg??.vga is drawn onto your 3D viewport every frame. The flat image in backg??.vga is usually then shaded to look like a ground, fading to black at the horizon. The sky here is drawn as a solid color. This solid color is taken from the very top line in the backg??.vga file. The whole sky, (the top half of the 3D viewport), is taken up by this solid color. You can have a solid black sky for nighttime scenarios, or a solid blue sky for daytime.

The pattern vibrates when the player moves to give the illusion of motion. If the pattern didn't vibrate 3D objects would appear to be moving relative to the ground, because the graininess of the ground doesn't actually move with respect to the player. Anyway, if you specify a 3D repeating floor texture with your floor icon, but no sky texture, then the backg??.vga image is flipped upside-down. The pattern on the bottom of the backg??.vga image is then put up above to simulate the sky or a high ceiling. If it is kept dark and properly shaded, this can make a very convincing ceiling. However, you will probably want to make a proper 3D textured ground panel, maybe something that looks like green grass, dirt, or leaves. Use GCSPAINT from the DOS prompt to edit the BACKG??.VGA file in your project directory. How do you know which backg??.vga file to change? When you set your level number, that number replaces the question marks. For example, if you had used the "SET LEVEL #" command from the MISC. MENU to set your level to #5, you must edit BACKG5.VGA to change your basic background.

Buildings

After making some artwork of buildings, use the world editor put up the building's walls. If you want the player to enter the buildings, make the insides of the buildings separate levels. When the player approaches a door, he'll cross a platform that teleports him to another level. This level will be indoors, although it will appear to have a similar size and layout to the building in your outside scenario. If you really want the user to go inside a building without changing levels, then it gets more complicated. The polygon floor sections don't work well as roofs for technical reasons.

Fences are easy and work well. Also, the floor polygons are good for making sections of road. Vehicles can be implemented as symmetrical objects, or even as complete constructions of windowed type objects. Telephone poles are very easy and realistic. Garbage cans and phone booths are also easy to make out of symmetrical objects.

If you put lots of pine trees in a level, you might want to select all of them and use the 'edit attributes' menu selection so you can walk through them. It can be pretty annoying to get stuck between trees.

You can connect outside levels in many ways. You must herd the player to where the outdoor levels connect. Put a perimeter

of window-wall type forest artwork panels bordering your level, then add one break where there is a level-switch platform. Or you can simply go in one side of a building, go down a hallway, and come out in another outdoor level.

EDITING YOUR 3D LEVELS

Moving/copying walls around

In the introduction to building a level, you read how to place walls and other objects. Doing a layout for a 3D game is more than just plopping down walls, however. It is also important to be able to edit your layouts, while deciding what to put into your levels.

Grid snap

In making a room, it is very important to get your walls panels to butt up against each other perfectly. Gaps even 1 cm wide can result in unsightly bright lines between walls. This is the reason for grid snap. When using grid snap, your walls will naturally line up along the grid. You don't have to worry about misalignment. The only problem with grid snap is putting a wall in a position that is halfway between the grid lines. Here, you need to go to the 'MISC' pull down menu, and use the command that aligns the grid to an object. This is very useful when working with walls of different sizes. Sometimes it is better to turn the grid off, and just use 'Object Snap', as described below. Also, when using very fine grids like 50 units or smaller, you will have to turn off 'Object Snap', or the two effects will interfere with each other.

Object snap

Object snap is very similar to grid snap. However, this feature snaps new objects to existing walls, rather than to a fixed grid. This is very handy when you need to join walls not aligned along a grid snap line. This may seem to make grid snap extraneous, but in reality, grid snap is better because with grid snap, you can always close your rooms and hallways. If you mix different sized wall sections, you can still manage to connect the walls with no gaps by using object snap instead of grid snap. However, you may have a problem when it comes time to close a room. When you mix wall sizes, you may find there is no wall section that will fit nicely in the leftover gap. If you use walls of 400 cm width and 400 cm grid spacing, you will never have this problem.

You can always turn both grid snap and object snap off, and place your walls free form. You can get quite close by operating all the way zoomed in. However, you will find that if you try to make rooms and hallways, your errors will pile up as you place long strings of walls. Eventually, you will be faced with some nasty gaps between your walls that are hard to fill.

Selecting objects

It is very common to put down walls that you decide to remove later. The process of changing what you have already put down is called editing. There are several things you can do to walls that have already been placed. These include moving, deleting, copying, rotating, and raising and lowering.

Like many Windows programs you may have used, these functions are implemented in a two-step process. First, you select which objects you want to operate on, and then you press the tool icon to make the action happen.

Select objects with the selection tool, which is the big arrow icon in the upper left corner of the screen. If you aren't in the middle of placing a wall, or doing something else, you are probably already in selection mode. When you click on existing walls or objects while you are in selection mode, red x's appear on those items. The little red x means that an object is selected. If you want to select a whole area of objects at once, you can either click on the center of each object, and 'x' them one by one. Or, you can click where there is no object, and 'rubber-band' a rectangle around a whole region.

This might work a little differently than other graphical programs you have used. In many other programs, clicking on a second object after already selecting another will cause the first one to become deselected. In the GCS, the more objects you click, the more are selected. Also, in many other programs, you hold the left button down and drag the mouse (while holding the button down) to make your selection rectangle. This is NOT the way the GCS works. To get the rectangle in the GCS, you simply left-click once in a spot where there are no close objects. A rectangle is started. Click again, at the opposite corner of the selection rectangle.

The real beauty of this selection system becomes evident when you want to pick and choose a few items that you DON'T want selected. If you left-click again on an object that already has a red x on it, the red x will vanish, thus showing the object as deselected. To move or copy nearly all the objects in a room, draw a box around the whole room to select everything. Then simply go through and deselect the few objects that you didn't want included.

A right-click will cancel all the selected items. Most of the editing tools will leave the items selected. The right mouse button is great for quickly deselecting, in case you don't have any further editing to do on those items. The tools that need selected items are the icons in the bottom 1/3 of the icon bar. The most commonly used icons are the move, copy, delete, and rotate icons.

Moving walls or whole rooms

To move some objects from one spot to another, first select the things you want to move. Then, click on the move icon. Now you will be asked to pick a reference point. Pick an endpoint of one wall that you have selected. The point you pick will be the 'drag point' of the cluster of objects that you selected. When you then click on the new grid position, the 'drag point' will be put exactly where the mouse is. Take, for example, the layout of a secretary's cubical in a large office type room. To move the secretary's cubical 400 cm down the wall, use the selection tool to draw a box around the desk, plant, partition, and wastebasket. Then click on the move icon. For a reference point, click on the point where the partition meets the office room wall. Then move the mouse pointer down 400 cm. Click on a different spot on the same wall. The GCS moves all the objects down, and puts

them in just the right spot relative to where the partition meets the wall.

If in the above example, I had chosen the center of his desk as a reference point, then it would have been harder to click in the right spot when choosing the move destination. I would be looking at the layout, trying to figure where the center of the desk would have to be to get the partition to butt up against the office wall perfectly. I could easily misjudge and wind up with the partition sticking through the wall, or leaving a gap between the wall and the partition. One problem you may encounter is that the grid does not let you move your objects exactly where you want. You can easily make the grid spacing finer with the grid icon tool. You can also turn grid snap off, but that might be undesirable when trying to match up walls. Of course, if you are just moving symmetrical objects, the grid is not really needed. If the grid is really preventing you from putting a wall just where you want it, the object snap feature should allow you to join walls perfectly even with the grid snap turned off.

How to copy groups of objects

The copy function is identical to the move function, except that the original objects stay in their original positions. A copy is placed at the new position.

How to erase groups of objects

The delete function should be obvious. Select whatever objects you want to discard. Hit the erase icon. All the objects that had the little red x's on them are gone forever.

Rotating groups of objects about a point

The rotate function is really fun to use. Select your objects and click on the rotate icon. Once you pick a rotation point, all the objects rotate counterclockwise 90 degrees. If wanted a different rotation angle, just click again to go another 90 degrees. To exit rotate mode, just click on the selection tool, or right click once. The objects remain selected after a rotation because you frequently want to move them after rotating. You can't rotate in 45 degree increments because that would involve changing the sizes of the wall sections. If you zoom in and measure the wall lengths on your screen, you will notice that the diagonal sections of your walls are longer than the right angle sections.

Altitude Adjustment

The elevation adjustment tool also operates on selected objects. You can raise a whole cluster of objects by a certain amount, or you can move them all to the same new elevation. Those two things may sound the same when you read the sentence, but they aren't.

If you wanted to raise the secretary's cubicle object cluster up by 50 cm, select all the stuff, and then click on the ELEVATION ADJUST ICON. Put a 50 in the text box, and click on the OK button. If you don't click on the ABSOLUTE button, the objects

would just move up 50 cm from their initial heights. This raising function preserves the fact that the wall phone on the partition was originally put at a higher elevation than the wastebasket. To move the objects back down, you would do the same steps, but you would type in an elevation of -50 cm. The negative number would move it back down.

Doing the same example, but clicking on the 'absolute' button, would have moved every object to the same height. The wall phone, the wall picture, the wastebasket, and the desktop would all be placed at the floor level.

Changing a placed wall into a door

To turn a wall panel into an opening door, select a wall, and press the door icon. You can actually make multiple doors at once by selecting several walls. Usually, though, you will want your doors to open in different directions, and to have different keys, etc. There is a whole chapter that goes into more detail about this tool.

The Find All function

When using the GCS, sometimes you will want to eliminate all instances of a particular wall. Instead of having to find them all one by one, the GCS can whip through your level and find all instances of a particular wall and select them for you. To use this, just select one wall object, and click on the 'Find All' menu command in the 'OBJECT' pull down. All objects of that type will be selected. An idiosyncrasy of this feature is that it only finds up/down copies of the object, or diagonals, not both at the same time. So in general, to delete all instances of a wall, you need to first find all the up/down and side to side walls and then delete them, and then find all the diagonal versions, and delete them.

Invert

Frequently it is necessary to flip a wall 180 degrees. Although you can do this with each wall individually with the rotate function, it is more convenient to select a whole bunch of walls, and press control-I.

Editing object attributes

Each wall in your level has various attributes. These attributes alter the way the wall is drawn by the 3D engine and how the wall interacts with the both the player and enemies. First select a wall. Then select "EDIT ATTRIBUTES" from the Objects pull down menu. You will see a list of settings. The default is for all these settings to be turned off. It is not advisable to set the attributes to groups of selected objects. This practice leads to unexpected errors and a million and one headaches. Select objects and set their attibutes on at a time and test your level frequently. Sure, this suggestion makes your level building slower and somewhat tedious, but you will save yourself a lot of frustration in the long run.

Don't Draw Backsides

Each wall has a front and a back side. The 3D engine does the 3D calucations to draw both sides of a wall. These calculations are a waste if the player would never have occasion to see the back of the wall. Thus, clicking on the Don't Draw Backsides button, speeds up the frame rate. When most GCS users put walls up, they don't pay attention to wall fronts and backs. Since both sides are drawn, the walls appear normally when you test your level. However, when you turn on the Don't Draw Backsides attribute, walls may appear invisible if their backs are facing the player. The walls are still there, but you can't see them. So, if you are using this feature and a wall is invisble, select the wall, use the invert command (ctrl-I) to flip the wall. Now the front of the wall will be facing you and it will reappear. Use this attribute with walls only. You will run into all kinds of critical errors if you use this feature with other kinds of objects.

Don't Fade With Distance

This attribute is used with lights, torches, glowing keys, and other objects that seem to have there own light sources. This attribute turns off fading and shading for the the particular object that you selected.

Don't Draw If Far Away

You can speed up the frame rate of your level by setting this attribute with small objects. Why slow down your computer by having it do the calculations necessary to draw objects that are too far away for you to see any way? This also applies to objects inside buildings or rooms that are way on the other side of your layout. This is really a feature that you should use only when you are tweaking your finished level to make it run at the optimal frame rate.

Can't Bump Into Object

By setting this attribute, you allow the player to pass through the object as if it were a ghost. This is great for making curtains, for example. This is a feature that you will need to use when making door frames, otherwise you won't be able to get through your doorways. You might also want to use this feature to cut down the frustration level of players who get stuck in dense areas of foliage or in rooms crowded with furniture.

Set Wall Lighting By Angle

You can enhance the 3D look of your level with this feature. Walls with a North/South orientation are shaded differently than walls with an East/West orientation. You'll see that corners look especially good.

Object Won't Show Up On Radar

You have to use this feature when constructing door frames that you want enemy characters to walk and shoot through. This feature is also useful for making extra secret rooms.

Set FTC On Solid Walls

This is used to create blocker panels. "FTC" stands for "Floor-To-Ceiling". This attribute tells the 3D engine to forget about drawing any object that would normally appear behind this wall. This is used to maximize the frame rate. Care should be taken when using this feature though. Never use this attribute with a windowed wall, fence, or any other wall panel that the player could look through, since everything behind the wall would be invisible.

ADDING DOORS TO YOUR GAME

How to Make a Door

Every good 3D game needs doors. Locked doors force players to search through your beautiful 3D worlds looking for keys. Other doors merely increase suspense about what might be behind the doors. Many layouts need doors as active parts of the scenery.

Doors are extremely easy to make. In short, the whole process consists of marking a normal wall section as a door, and then setting a few parameters.

Look at your layout and decide where you might want to place a door. Then click on the brick wall icon, (or the windowed brick wall icon), and place a wall section where you want your door. It doesn't have to look like a door. You are free to make 'secret' doors out of any solid wall, or windowed wall object. You cannot make doors out of symmetrical objects like trees, however. After you have placed your wall object, it is just another immobile wall section.

After placing the object, return to selection mode by right clicking on your mouse button, or by left-clicking on the selection tool icon. Then left click once on the center of the door-to-be. When the little red 'x' is on the door, click on the door icon on the tool bar.

A door menu comes up with some text boxes and push buttons. Fill in the numbers, or use the defaults. Push the buttons to choose the sliding direction, and then push the OK button with a mouse click. Your door should now be functional!

NOTE: Doors will fail if you try to make them too wide. Walls greater than 600 cm wide may fail, and walls 1200 cm or wider will certainly fail when you test your level.

Now we can go into detail about the number parameters.

Resistance to forced entry

This number determines how hard it is to break the door. A value of 0 means that merely bumping the door breaks it, and it then stays open forever. Putting a 1 in here makes it open by shooting or kicking. Increasing this number increases the difficulty

of breaking the door. If the door resistance is too high, the door is not affected by even the player's most violent attacks.

For example, if the player bumps the door while shooting his weapon, the door will only be weakened by repeated attacks if the resistance is lower than eight.

If the door is set to resistance 31, the door is only damageable by jump-kicking. This takes some practice by the player, but it is a fun action master. Please note that on levels where you have a texture-mapped ceiling, the player cannot jump for technical reasons. Therefore it is best to leave doors like that for outside areas or for those indoor areas with high ceilings. Doors with resistance set higher than 31 will not be affected by the player at all.

Number of Seconds to Stay Open

When the door is opened by enemies, or by the player, it stays open a while before closing. Doors will not close on the player. They will stay open until the player gets about 800 cm away.

Key Value

The key value number selects which key will be needed to unlock the door. The player must have this key in his inventory to get through. If the player does not have the right key, an icon of the needed key will show up in an inventory window on the game play screen. The keys are coded by color. If you have changed the default inventory item pictures, then these colors might not be accurate. If you are modifying the picture images for the keys, it is up to you to keep track of what color keys you have modified.

Open upon Explosion

The button on the door dialog box labeled 'Open on Explosion' controls makes the door open if an explosion from a hand grenade, or other source, occurs close to the door.

Tips on placing doors

All doors defined in this manner are sliding doors. It is possible to make doors that rotate open, or slide up or down by defining an animated object. This discussion is limited to normal sliding doors, however.

When these doors slide open, the artwork panel merely slides in one of four directions: North, East, South, or West. The actual orientation of wall in the 3D game does not affect the direction of the sliding motion. Therefore, it is easy to make a door that slides in or out, like a Wolfenstein secret door. It is even possible to make a diagonal wall slide, which can be used for a very cool secret door.

Door Frames

You need to set certain object attributes to make your door frames operate properly. First, select your door frames. Then, select "EDIT ATTRIBUTES" from the Object pull down menu. Click on the "Can't Bump Into Object" button. This allows the player to pass through the door frame. Next, click on the "Object Won't Show Up On Radar" button. This allows enemy characters to pass through the doors. It also allows enemy characters to shoot through open door ways.

How to avoid graphical problems with doors

The normal sideways sliding doors can get you into trouble, however. When the artwork slides to the left or right, the drawing of the wall section does not stop at the door frame. Commonly, GCS users put doors in places where the door panel comes out through the side of a wall when it opens. As game designer, you are responsible for making sure the door panel is hidden when it slides sideways. The easiest way to do this is to put your doors in hallways. Make sure that the hallway has some unused room on the side for the door panel to slide into. Otherwise, the player may be in a room at a time when an enemy unit opens the door, and the door panel will slide through the supposedly solid wall of the room.

Another problem that can frequently happen is that the door slides out along another wall panel lengthwise. Here, the door and the wall panel are exactly overlaid in the 3D world. The graphics then breaks down and you will see a shimmering mess of vertical strips. This is usually easily avoided by moving your door into a door frame by a little bit. By door frame, we mean that you build a square of dead space on either side of the door. Then you place your door at the entrance way. However, if you set the object to be a door, and leave it here, you will run into the same effect described above. Offset the door 200 cm, so it will not be aligned with any other walls when the panel opens. The snap-to-grid will make this offsetting impossible if the grid is set to the standard 400 cm. You will have to change the resolution of the grid or turn the grid feature off.

So click on the grid tool and change the grid setting to 200 cm. Then select the door with the selection tool, and click on the move icon. Move the door one 200 cm grid line back, so that the door is now centered in the frame. Now when the door opens, the panel will slide safely into the dead space. Of course, you can set your grid spacing to as little as 50 cm to inset the door just enough to keep the 'scalloping' from occurring. It's best to be zoomed in before setting your grid to such a fine spacing.

If you make a door, but you can't go through it when you test your level, read the section above on Door Frames. Likewise, if enemies won't walk through or shoot through your door, the section on Door Frames will explain the attributes that must be set to make the door function properly.

UNIVERSE REGISTERS

In the Pie 3D GCS, it is possible to make an object blow up when you pick up a key. It is even possible for just about any animated object to respond to any other animated object. Universe registers make this a possibility.

Universe registers are holding places for numbers. Animated objects and platform objects can put numbers in these holding

places. In addition, animated objects can look at the numbers in these holding places. There are 256 universe registers, and they can be set to any value between 0 and 255. There is an animation command to set these registers. Registers can be set by platforms and inventory items too.

For example, we can set a platform object to set universe register 12 to value 1 when the player walks down a hallway to a room. Maybe the walls of the hallway are not actually regular walls, but actually animated objects, that happen to look like wall objects. Each of these animated objects has been in a loop waiting for universe register 12 to be set to one. When this happens, the walls roll up. Enemies attack. We've just used universe registers to communicate to many animated objects, thus setting a trap for the player.

Using universe registers and animated objects, it becomes possible for objects to have complicated relationships. It is possible to make an animated object explode if the player shoots at a piece of equipment, has the red key, and has NOT explored a certain hallway. The mechanics of how to set and test universe registers is in the chapters about animated objects, platforms, and inventory items.

Special purpose registers

Not all universe registers are the same. In fact, universe registers 128 through 168 are reserved for other functions such as keeping track of the number of enemies, and the lighting status of the various levels. Also, universe registers 122 through 127 are special purpose registers. All the other universe registers not mentioned here are general purpose registers, and are available for your use. Don't use register 0, however, since not all commands will work with universe register 0.

Registers 122 through 126 are reset during game play. You can use the animated object commands to test these universe registers, but trying to put your own numbers in these particular registers will not work. The following table explains what values will be in these registers:

122 animation frames per second of game
123 current player health 100=perfect
124 number of hits scored on opponents and animated objects
125 number of ammo rounds fired by player
126 number of enemies killed

Register 127 is the most special of the universe registers. When an animation object puts a non-zero value in this register, the game is ended. If a number value smaller than 128 is put in universe register 127, it is assumed that the player has been killed. This means that the screen will fade to red before exiting the game. If a greater number than 128 is put in universe register 127, the game will just end, and the menu program will put up the appropriate screen to show the user. For example, if the user takes too much time to save the princess, an animated object could put a value of 130 into universe register 127. The game would end with value 130 passed to your menu. You would then have the menu put up a picture of the princess's demise, and an text

message, "Too late! The princess has been devoured by the deadly brine shrimp!".

You could also have an animated wall that has electric spark artwork on it. If players get too close, the animation object could put a 19 in universe register 127. The screen would turn red, signifying death. Then the menu program could pick up that 19, and explain to the player that he had been electrocuted.

OBJECT LIBRARIES

What is an object library?

Whenever you click on the brick wall icons, or the tree icon to place an object in your level, you are using the object library. The list you see is just a list of objects in your current object library. You will notice that each wall has a horizontal and vertical size. This is all made possible by the object libraries.

The Pie 3D GCS allows you to store all your objects in libraries. This is much more sensible that just having a huge directory filled with .vgr artwork files. Why? The library directories live inside a master palette directory. You know all the artwork in that library will be made for that particular set of colors.

Another reason for object libraries is that each object has its own 3D world size. If you had to tell the GCS the size of a wall every time you wanted to place it, you would go bananas.

In addition, the object libraries allow you to organize all your artwork into separate directories. You might have a library called castle with all artwork for doing the insides of castles. Maybe another library could be called forest. This one would have lots of trees in it, maybe some squirrels, etc.

Also, you can purchase more object libraries from Pie in the Sky, or you can trade some with other GCS users. The object libraries can be easily copied and added. The only thing to watch out for is to put your new object library in the right palette directory!

Choosing an object library

Object libraries are not like palettes. In the GCS, when you choose a palette for your new game, you are stuck with that choice forever. However, you can keep switching libraries during the construction of a level with no limitations. You can even start your own object library with no trouble at all. Any time you are in selection mode, you can click on the file menu choice called 'OPEN NEW OBJECT LIBRARY'. You will get a standard dialog box asking for a library name. You can type the name, or choose from the list of existing ones. If you type a name that doesn't already exist, then you can create a new one.

If you create a new one, it will create a directory inside the current palette directory. Right after you create it, it will, of course, be empty. You will need to use GCSPAINT to start creating artwork and saving it into the new library directory, or else you can copy .vgr files in from other sources into the directory.

Importing or creating new artwork

If you have some artwork that you would like to put into your game, you have to get it into an object library before it can become available when you click on the wall icons. Importing or creating artwork is a two-step process. First you must create a .vgr image file and get it into the object library directory on your hard disk. One way to do this is to choose the library directory you want (with the FILE menu command called 'OPEN NEW OBJECT LIBRARY'), and after doing that, click on the GCSPAINT icon. This is the one with the paint brush. You will be popped right into the paint program with a new image. You can create artwork there, or load a .gif, .pcx, or .bmp file, and then save it.

Now you have done the first step. But, it is not enough to get the object to appear on the list of available items. To make the object usable, you must set a size for it in the 3D world. Once you have done that, your new items will be ready to place in any level of your game.

To finish adding your image to the object library, go to the 'OBJECTS' pull down menu on the top of the screen. Click on 'Library Import/Modify.' Then the GCS will ask you what kind of object you want to import. The first three choices, all expect the same kind of .vgr file to be in the object directory. You can choose 'Solid Walls' to add a new wall. Next you will be asked if want to import and object, remove an object, or modify an existing one. You want to import. If you get a red error box saying 'No Files Found', then you have not saved your new .vgr images in the right hard disk directory. If there are some .vgr files there, you will get a scroll box that will let you select the image that you want to be available to GCS as an object.

As you move up and down the list, you will see a tiny picture of what the wall looks like. Click on the one you want. Then a box called 'Set Object Size' pops up. Set the size you want for your wall. The overwhelming majority of walls are 400x400 centimeters. This is a convenient size to work with. Sometimes you want walls that are 800x400 so you can make a larger maze with fewer maze objects. See the chapter on layouts for more information about wall size choices.

If you are defining a symmetrical object like a tree or a floor lamp, there is no advantage to keeping them a standard size like 400x400 cm. Do remember that the game engine will stretch your picture to the size you specify. See the chapter on image artwork for a discussion of that stretching. Once you click on the OK box, you have set the size of your image. The next time you click on the brick wall icon to place a wall, your new object will there for you.

There is no reason why you have to have only one size for an image. You can go through the same steps again, but this time, specify a different size. The library can have many different sizes of the same piece of artwork. It may get a bit confusing when you have to select among objects of the same name when you are placing, but the size is always there so you can tell them apart.

Also keep in mind that the little view window stretches the image file to fit in the little box. So very tall thin things get stretched way out like a fun house mirror. Sometimes it makes it hard to identify the objects. Although you may have given this object a different size, it will always look the same in the little preview box.

Modifying an object in an object library

Modifying an object that you have already placed into the object library is easy. If you want to change the picture, then just click on the GCSPAINT icon, and change the .vgr file with the paint program. Your changes will be reflected, even if you already placed the object before you made the changes. Changing the size is easy too, although the size of all the objects in your levels that you have already placed will NOT be resized. The new size will only apply to objects that you place after the change.

Removing an object from an object library.

This is very straightforward. Choose an object and click on it to remove it from the library. This will NOT remove any instances of this object from your levels, however. To accomplish this action, you will have to delete those objects from the levels in the normal fashion. When deleting an object from the library, the .vgr file is NOT erased. The only way to get rid of that .vgr file is to use the DOS prompt del command.

Adding Animated objects to an object library See the chapter on animated objects for more information about this.

TEXT MESSAGES FOR THE PLAYER

Pop up text messages allow a friendly character say something to the player. Also, the player can read messages left on the wall or floor. In other situations, a message informing the player of some event can be appropriate. It is easy to create text messages that will be put up on the screen for the player to read. Animated objects can be made to put up messages, so can inventory objects like memos.

You may enter as many as eighty messages into a file that is read later by the game engine. The messages can have up to thirty-eight characters per line, up to ten lines. You assign a number to each message when you write it. Later, when you are making your inventory item or animation object, you give the command to write that particular message number.

Do not exceed the maximum number of characters on a line, or the maximum of lines per message. If you do, the text will write outside its text box, and leave a mess on the player's screen. Remember not to leave extra spaces on the ends of your lines, because spaces count just the same as other characters. Also, blank lines in your messages count too. If you need a handy way to make sure you have 38 characters or less on a line, temporarily type this as the first line in your message:

01234567890123456789012345678901234567

Then don't make any lines longer than the above. When you are finished with your message, delete the line so it doesn't appear in the final game.

Editing the pstr.txt file

For example, let's say you want a friendly man to wave the player down to warn him that there is a trap behind the red door. Edit the file pstr.txt in your project directory with your favorite text editor, DOS EDIT will do fine. Insert the following message in the pstr.txt file:

```
string 3
Hey pal, don't tell the snake men I told
you, but they have set up a trap behind
the red door for you.
sdone
```

Compiling your messages with pstr.exe

Run the text file compiler program using the command line: ..\pstr

!!!! NOTE !!!!! This will NOT work unless you put the two dots and the backslash '\' before the word 'pstr'. Also, make sure you are in your Project directory which is a subdirectory of the ENGINE directory in your P3DGCS directory.

Go back into the GCS program to make your animated object. Make some nice pictures of a guy waving, and a guy standing with his arms down.

Make an animated object using these artwork frames of the friendly guy. The object should alternate between two frames of the guy waving his hand. Then check to see if the player is within 400 cm of the animated object. If yes, give the animation command to put up text message #3. The message pops up, just as if the friendly guy waved the player down and talked to him.

To make it more concrete, here is an example. Say you have two images of a guy waving his hand, guyhand1.vgr, and guyhand2.vgr. You put both these .vgr files in your library directory and import them so they show up on the list of symmetrical objects when you click on the tree icon.

Create a .vgr image of the guy standing there, and call it guystand.vgr, and import it also. Then you go back to the library, and chose animated object, and then design/create. You say no to the 'set position' question, and then set up the animated object with the following frames:

FRAME #0 guyhand1.vgr no-operation (delay 2)

```
FRAME #1  guyhand2.vgr  skip if player close  (400 units)
FRAME #2  guyhand2.vgr  unconditional branch to #0
FRAME #3  guystand.vgr  no-operation  ( delay 6 )
FRAME #4  guystand.vgr  display text message #3
FRAME #5  guystand.vgr  unconditional branch to #5
```

This guy will stand there waving until the player gets close. He will put his arm down for a moment, and then he will put up the text box. It is up to you to type the message in PSTR.TXT, and use PSTR.EXE to compile the text message before you try this out. If you forget to define the text message, you will get a critical error when the computer tries to put up the text message that isn't there.

The pstr.txt file in your project directory is where you enter the text for the messages. It is a little tricky to get used to the file format for the messages. Each message starts with a line that has nothing on it but the word 'string', a space, and then a decimal number. The word 'string' must be in lowercase characters.

After this line, enter your text. No line may be over 38 characters long, and there can be no more than ten lines. If either of these rules is violated, the text will appear as mishmash when the messages are put up. You don't need to have ten lines every time. You are free to use any number of lines between 1 and 10. After your message lines, there must be a single line with nothing on it but one word: 'sdone'. This tells the program that the message is over. After the last 'sdone' command in your pstr.txt file, there should be one line with nothing on it but the word 'end'. This tells the program that there are no more messages after this line. This line is important.

It is important to recognize that many editors allow you to edit text and save it as an ASCII or TEXT file. Many editors will insert strange characters, or put all text on one line when you save the file. It is important to use an editor that makes a pure DOS text file with carriage returns and linefeed characters. To test your text file, use the DOS TYPE command to type your message file on the screen. There should be no lines longer than forty characters, and the 'string' and 'sdone' lines should have no other letters, words, or spaces on them. For example, this is how it should look when you type out your pstr.txt file with the DOS TYPE command:

```
string 0
NOTICE TO STAFF:
These emergency doors have been
automatically extended, and cannot
be reopened.
sdone
```

Getting more specific about pstr.exe

Now that you have your pstr.txt file done, it is time to translate the messages into a form that the game engine can read. This

file is called pstr.str. It is created automatically by the pstr.exe program. The pstr.str file must be in your project directory to work with the game engine. You need to run the program pstr.exe from the dos command line. Because the pstr.exe program is in the ENGINE directory, not your project directory, you must call up the pstr.exe program from one directory back by using the following syntax:

..\pstr

The '..\' characters tell DOS that the pstr.exe program is located in the ENGINE directory instead of the project directory. It is equivalent to the line:

c:\p3dgcs\engine\pstr

Assuming you have installed the GCS in the directory 'c:\p3dgcs', the above line would have the same effect. Anyway, when the pstr.exe program is finished running, it puts a message up like:

String data takes up 24520 bytes

Don't worry if your number is drastically different. It really depends on the number and size of the messages.

Entering your messages in the pstr.txt file, and creating the final pstr.str file in the proper directory requires some computer experience. So if this set of steps seemed daunting, you can at least be relieved that no other part of the GCS needs such manual DOS prompt manipulation.

PUTTING ENEMY CHARACTERS IN YOUR GAME

No 3D action game is complete without enemies. You can place up to 32 enemies in a level. Placing them is as easy as placing trees, but with more options. To place a guy, click on the icon that looks like a guy. Up pops a big dialog box filled with options.

First, you must determine the properties of the enemy unit. These parameters adjust their strength and behavior.

Quickness This determines the quickness of the enemy's movements.

Step size This determines how fast the enemy moves by adjusting the distance traveled in each walking step.

Resistance This sets how much damage the enemy can sustain before succumbing.

Attack Strength This controls how much damage is inflicted on the player when the enemy strikes or fires upon the player.

Attack Accuracy How often the enemy will hit you with his weapon.

Range How close an enemy must be before his attacks harm the player.

Enemy Character Behavior:

Sentry, then hunt The enemy stands still until he sees the player, then chases after the player in attack mode.

Random Patrol The enemy walks about at random and then chases the player when he or she is finally seen.

Stand and shoot The enemy stands still, but if he sees the player, he will remain in current position, but will shoot.

Enemy Artificial Intelligence:

Einstein with an Attitude The enemy is highly alert

Average but armed The enemy is somewhat alert

Dufus The enemy is not very alert.

What was the question? The enemy is so oblivious, he will almost never attack a non-shooting player.

The basic behavior of the enemies is always the same. They ignore the player until alarmed, and then they hunt the player down relentlessly. What causes them to become alarmed is a complex calculation.

What alerts an enemy?

Initially, all enemies are in casual state. The only event which can cause enemies to become alarmed is seeing the player. However, not all enemies will become alarmed right away upon sighting the player. Whether or not they recognize the player as hostile depends on the intelligence setting, the viewing angle, and what the player is doing.

The most intelligent enemy units will recognize the player on sight from any angle, and become alarmed. The least intelligent will not become alarmed even if the player is standing in front of him, blasting his pal. All enemy units will become alarmed if they take a hit from the player, however.

The view angle is very important in being recognized. If the player is directly behind an enemy, the chance of alarm is much

less. Also, for the most part the enemies can't see through walls.

If an enemy comes near a slain fellow enemy, alertness can also be raised.

What the player is doing affects The chance of being detected depends on the player's actions. Certainly, if the player is running around shooting weapons, he is much more likely to be spotted than if he is standing still. All this info about the player's running, jumping, or shooting is boiled down into one number called the profile.

The bigger the profile, the easier the player is to spot. Running or moving quickly will raise the profile. Shooting a weapon raises the profile drastically. Being wounded will also raise the profile.

Enemy character sounds

When an enemy first becomes alarmed, he will play one of the 'HEY!' sounds. These are sound numbers 20-23. The sound depends partially on randomness, and partially on 'round-robin' selection. For more information about setting sounds, see the chapter on sounds.

How enemies behave

Enemies shoot when they are alarmed. If the player is out of range, enemies will not continue shooting, but will serpentine towards the player. At the halfway point, enemies will stop and take at least one shot. If within range, enemies will shoot repeatedly before venturing closer.

These steps will be repeated until the enemies are so close that there is no point in coming any closer. Then they will just stand and attack continuously.

If an enemy gets hit by the player's weapon, the enemy's shooting is interrupted for a moment. How much damage is done to the enemy is dependent on many factors. The range and accuracy of the player's weapon are very important. These parameters are adjustable by editing the text file 'weapons.txt' in the engine directory.

In addition, the resistance to damage setting determines how much damage is absorbed by the enemy unit when hit by the player's weapons. If the enemy is damaged enough, he will attempt a retreat. If he runs into a wall as he flees, he will turn and fight to the death.

If the enemy actually gets very far away from the player after running away, he may return to the unalarmed mode, but this is very infrequent.

Making your own enemies

The Pie 3D engine was designed for 3D action games against human enemies. Hence it is easy to create games that have human enemies with guns. Two kinds of enemy soldiers are available with the basic GCS package. Although they behave identically, they have different appearances. The customization of the guards that is available is the changing of the appearance.

There is a subdirectory in the main p3d directory called enemies. In this directory are many .vga files which are pictures of the enemies from various angles. Theoretically, there should be eight frames of artwork for every possible position of the character. For example, there are eight frames of walking with a right foot forward, eight frames with a left foot forward, eight more frames with gun being drawn, eight more with a gun out, etc.

Thirty-four possible unique enemy positions, means 272 pieces of artwork! When working within the DOS 640k limit, this is impossible. So, there are tricks to reduce the number of frames needed.

The first trick is to make the enemies symmetrical. Looking at the enemy from the left side is the same picture as looking at him from the right side. The only difference is that the picture is reversed left to right. So instead of having to store the same image pixels in memory twice, just tell the 3D engine to use the same artwork image, but to draw it backwards.

The second trick is to take advantage of the fact that in certain situations the enemy will always be facing the player. The enemies will not fire their weapon unless they are facing the player. Therefore there is no reason to have artwork frames of those who are facing different directions. Therefore, for the attack frames, you don't need 8 views of each animation position. For the attack frames, you only do the head-on view.

Likewise, the tear-gas and the immobilized frames only have one view. Presently the death sequence can be done in front-view only, or you can have all eight views.

How to modify the appearance of the enemies

Each set of original enemy artwork is kept in a separate directory on your hard drive. All the object libraries and enemy directories are specific to a certain palette, or color set. If you tried to mix images from different palettes, the colors come out all wrong. The enemies, walls, or whatever will look psychedelic.

This is not a limitation imposed by the Pie 3D game engine. VGA 256 color mode can only put 256 unique colors on the screen simultaneously.

There is a separate master directory for each palette. The enemy directories and object libraries are all kept in directories that are inside the master palette directory on your hard drive.

Since your game uses a set palette, you can use only the enemy directories that are in the palette subdirectory you are using. This sounds complicated, but it is really very simple. Say you installed the GCS into the directory c:\p3dgcs. Also, let's say you chose

to use the palette called "$rp9a." All the artwork, weapons, inventory items, and enemy files use the palette called "$rp9a." The way the GCS works, for each palette there is a master directory. Thus, there is a directory in the c:\p3dgcs directory called:

$RP9A\ or the full path name is: c:\p3dgcs\$RP9A\

All the libraries, weapons, inventory items, and enemy artwork files that you can use in your game must be taken from this def_enem directory.

To keep all those libraries, weapon files, inventory items, and enemy images organized, they are all kept in separate subdirectories within the master palette directory. So the enemies that are available in the palette you have chosen for your game, might be in a directory called def_enem\. Now this def_enem\ directory must be located within your palette directory $RP9A\, which is in your GCS main directory.

The basic enemy is a human with a handgun. So if you wanted to edit those images you would want to go into the directory:

c:\p3dgcs\$RP9A\def_enem\

The easiest way to modify the guard appearances, is to enter the ENEMIES DIRECTORY, and use GCSPAINT to change each artwork image. So if you want to change the guys into Barney the purple dinosaur, you would bring up GCSPAINT, and choose the save/load menu, and load .vgr image. Click on the double-dots filename. This will put you in the main palette directory. Then choose the ENEMIES DIRECTORY you want to modify. The first time you try this, you should probably choose a directory named 'custom1', so that you don't muck up the original def_enem directory. You can make a copy of the default enemy directory simply by selecting 'Create Directory' from the FILE menu. Then select 'Create Enemy Directory'. The GCS will then create a new directory for you, complete with the bad guys palette-matched to use the same 256 colors you use in your game.

Just bring up each .vgr file in that directory. Change the picture from a gun-toting human to a purple dinosaur picture. Make sure you replace each image with Barney doing the same thing that the bad guy in the original image was doing. If your first image is the front view of a guy with his left foot forward, then you have to replace it with a picture of Barney from front view, with his left foot forward.

This method is the easiest, because you don't have to decipher the filename or edit any text files. However, you must stick pretty close to the limitations in the original. This means, that Barney must remain symmetrical, and only have those certain frames for the death sequence, etc. If you want to have varying resolutions of the various frames that make up your Barney, or you want to make an asymmetric Barney, you would then have to edit the text definition files. However, you must read the comments in the text files carefully before you make changes, because it is very easy to make changes to these file that make critical errors occur in your game. How to change these definition files is not covered in the manual because it is not recommended for most game builders. However, we decided to leave the flexibility for those that really want to get into the nuts and bolts.

WEAPONS

What would a 3D action game be without weapons? Whether it is a magical candy cane that turns pine trees into xmas trees, or a rocket launcher that blows your enemies to pieces, weapons are a necessity.

How weapons work

Weapons in the Pie 3D game engine are very much like other animated objects. Controlled by an animation file, each frame of the weapon is a different graphic object that must be loaded into memory. The animation file controls the placement of each animation frame on the screen, and controls what happens when the user pulls the trigger, etc.

Besides that animation file, there is a text file called weapons.txt which controls the capabilities of the weapon. By editing this text file, you can set a weapon's killing power, required accuracy, and range. You can require the user to be holding that particular weapon in his inventory. In addition, this text file is also where you set the weapon display name.

A weapon set consists of the .vgr images for the weapon artwork, the animation text file and the weapons.txt file. This is what controls the weapon's attributes and game play properties. Whole weapon sets are kept in directories on your hard drive.

For example, for an action shoot 'em up game, the standard shotgun, machine gun, and other weapons might be in a directory called:

stndweap\

Change to that directory from the DOS prompt, and use DIR to get a directory listing. You see files like shotgun1.vgr etc., plus fldshtgn.txt, and weapons.txt. Together, this would comprise an entire weapons directory.

Each weapon directory must be located in a master palette directory. This is because all the artwork used must use the same 256 colors as the rest of your game. If you try to use artwork made with a different palette, all the colors will be wrong, and your weapons will look like color negatives.

Customizing the weapons

If you want to customize the weapons, you should probably start by modifying the standard ones. Use the 'CREATE WEAPON DIRECTORY' command from the file menu, to create a new weapons directory in your master palette directory. This will copy the default weapons, (shotgun, machine gun... etc.), into a fresh weapons directory. The GCS will match the colors in your default weapons to your current palette. If the color palette you are using is very different from the basic palette of minfiles\weapons directory, some weapon artwork may suffer in the translation. But, at least you would have a starting point for

modifying the existing weapon artwork.

By far the easiest method to customizing your weapons will be to edit the existing weapons pictures. If you make your weapons similar in size and action, you can use GCSPAINT to change each weapon to suit your game.

Just bring up GCSPAINT like you are going to create or change a typical .vgr wall panel. Then click on 'load' to load a .vgr image. However, instead of choosing a .vgr image in the current library image like you normally do, click on the two red dots listed along with the .vgr filenames. This will take you into the main palette directory. There probably won't be any .vgr files in this main palette directory, since normally the main palette directory holds only one palette file, and a bunch of subdirectories.

Then click on your new weapons directory name, (the name of the weapons directory you had previously created with the 'CREATE WEAPONS DIRECTORY' command.) Now that you have switched GCSPAINT to the right directory, there should be lots of .VGR artwork files to choose from. Load a few of them. Examine them to give you a feel for the filenames, the weapons, and the dimensions of the raw images. If you start to change the images, remember that your job will be MUCH easier if you don't change the size of the original files. Try to keep the approximate shape and size of the original weapon you are modifying.

You might want to tweak the weapons.txt file that is in that directory with a text editor. This would be to change the range, power or accuracy of the weapon, or to change its default sound effect. If you want fewer than nine weapons, you may also want to make a smaller weapons.txt to reduce the number of weapons. Of course, if you don't place that weapon on your levels, then the player could never pick up that weapon anyway. So, you won't have to mess with the weapons.txt file.

The .vga files in your weapons directories that contain the letters 'fat' are usually the pictures that appear on the control panel.. These pictures display the current weapon selection. They usually have some text describing the weapon, so the player doesn't have to recognize the weapon by drawing alone. These files are .vga files, and hence they cannot be saved when calling GCSPAINT from the icon in the GCS program. To edit them, you will need to exit from the GCS, and run the paint program manually from the DOS command line. The images you save will be in the .vga format. As a rule, if you need to edit .vga files, call GCSPAINT from the DOS command line.

If you really want to make very different weapons...

If you want to make a weapon that can't be done by just changing artwork frames on an existing weapon, then you have your work cut out for you. You will need to work out your own weapon artwork from scratch, and figure out size definitions.

Make sure that none of your weapons .vgr files are bigger than 16000 bytes because the weapon bitmaps are stored in EMS/XMS memory.

You will need to modify the file objweap.txt to define the horizontal and vertical 3D world sizes of the weapon. Edit the

animation text file for that weapon also. The default weapons animation files all start with the letters fld????.txt. These are the files that coordinate which weapon frame gets shown at what position, etc. There will be comments in those text files to help you decipher the correct way to do it.

Of course, you will want to edit the weapons.txt file to give your weapons a specified range and power. There are some comments on how to do this in the text files.

Go into your weapons directory. If you read weapdef.txt, pickups.txt, and weapons.txt, you can make your own custom weapons from scratch. We will warn you that it is not easy.

Do not start modifying a weapon set that is already in use by other projects. It is really easy to create a new weapons set simply by clicking on the FILE pull down menu. Make modifications in your new weapons directory.

You should read the chapter on inventory items before delving deeply into custom weapons. That chapter goes into detail about the concepts involved in making your own weapons. In addition, you will need to make a new inventory object for your new weapon.

You will need to read fluidmg.txt to learn how to actually specify the animation.

Keep in mind that every level within your project MUST use the same weapons directory. This is because you can carry weapons and other objects from level to level. If you make a mistake, and some levels were set to use different weapons directories, you will get critical errors. This would not affect the 'test level' feature, since you cannot travel from one level to another in this mode. This would only occur in your final game.

MAKING ANIMATED OBJECTS

This feature of the Pie 3D game engine is essential when making 3D games. You can make birds fly, operate the lights, determine the fate of the player, play sounds, or teleport the player.

Animated objects are like shape-shifters (like Odo in the FOX television program 'Deep Space Nine'). These objects can become entirely different things from moment to moment. It is even possible to start as a wall panel, turn into a tree for a moment, and then become invisible. Besides being able to change their appearance, animated objects can move on preprogrammed paths. They can also make simple decisions, have multiple behaviors, and change appearances depending on events that occur during the game.

The most complicated, and the most powerful, feature of animated objects is the command capability. Within each frame you can have the object do any number of tasks. One example is the 'unconditional branch'. This just means that instead of going to

the next frame, the animation will jump to a different frame number. In addition there are animation commands to play sound effects, damage the player, or make text messages appear on the screen.

There are also commands which allow animated objects to set or test universe registers. Since all animated objects can read and write numbers to these registers and take actions accordingly, animated objects have the power to actually communicate with each other, and respond to various events in the game. To learn more about universe registers, see the chapter on universe registers elsewhere in the manual.

How to make an animated object

To make an animated object, bring down the OBJECTS MENU, and select the LIBRARY IMPORT/MODIFY function menu. A menu pops up asking what type of object you are interested in. Choose ANIMATED OBJECT. Another menu asks if you want to import a new animated object, modify an existing one, or remove one from the library, or design a new one. Choose DESIGN NEW ANIMATION. First, a box pops up asking if you want to set the frame position. If you want to make a stationary object, one that doesn't move around, say no. If you want to make an animated object, one that moves, press yes. We recommend that you say no for most objects, because setting position for each frame is complicated and hard to do.

A big scroll box comes up with a big list of animation frames. In the leftmost column is the frame number. Just like in a movie film, each picture in the film has a frame number starting from frame number 0 at the beginning. The next column is a name of the image. This can be a solid wall, a windowed wall, or a symmetrical object. The next column is the command.

The first thing to do is to move the mouse up to the entry in the scroll box for frame # 0, and then click. That brings up a dialog box for the various options for frame # 0. You must press the 'set object' button to select the appearance for frame #0. The program responds by letting you choose from a list of objects. This will set the appearance of the animated object for frame #0 only. In most animated objects, all symmetrical objects are used. This is because animated walls aren't usually needed. However, there are applications for them, like doors that respond to the player. Don't be alarmed that your frame #0 line in the big scroll box still says 'NONE' for the object picture. The scroll box will not get updated until you click on 'DONE WITH THIS FRAME.'

If you are doing a moving object, you must set the position for each frame. This gets a little tricky. When you click on this, you will go into 'place' mode. However, you are not really placing items into your level, you are only specifying where you want your animated object to move. After picking the point for each frame, you will be prompted for an elevation. This is in case you want to raise your object off the ground.

Remember that you are not placing anything in your 3D level when you are picking these animation positions. When you are finished defining this animation object, do not expect it to be in your level. You must explicitly place it into your level by clicking on the FILM ICON. When you are defining an animated object, you are making a reusable object. This object can be put in any level in any project that uses the same main palette directory. For example, let's say you wanted to make an animated rat,

and you wanted 30 rats in your level. You don't want to have to define 30 unique animated objects. Make one rat and then place that rat 30 times in various spots in your level. That is why we don't automatically put the animated object into your level when you are finished defining it.

If you need to scroll or zoom or use the arrow keys while you are in place mode. Click on the MAGNIFYING GLASS ICONS, do your zooming or scrolling, then right click to return to your placing. Unfortunately, you will lose the grid. It will still be active, but it will be unseen.

Now it is time to choose a command. Click on the set operation button. A large menu will pop up with various command buttons. Choose the command you want. For now, just click on 'NO OPERATION'. The other commands will be discussed later. After you click on the 'NO OPERATION' icon, a box comes up prompting you to set the delay. A delay of zero is the fastest, but is generally not recommended because then your animated object will be dependent on CPU speed. For example, the animated objects will run faster or slower depending on how fast the player's computer is, and how fast the game is running on that particular level at that particular moment. For fast motion, choose a delay of 1. For most normal speed animation, choose a delay of 2. To wait several seconds before going to the next frame, choose a delay of 30 or 50. Do not specify a delay greater than 109, or your delay may not work. If you want a longer delay than that, just put down identical frames with delays of 109.

Now you have finished specifying that animation frame. You should now click on 'DONE WITH THIS FRAME.' Don't select 'INSERT FRAME', since that is for something else.

Now you are back to the scroll box, and it is updated to reflect your command and picture. You then can pick another frame to specify. It makes sense to go down the list of frames one by one. Don't leave blank frames between your used frames, or your animated object may not work properly.

You may have up to 40 animation frames in an animated object. Most objects use about 10 to 20 frames. Usually the object loops to the beginning rather than running through all 40 frames. For example, don't feel like you have to specify all 40 frames. For example, a flickering flame might just have three different pictures of flame, and after the third frame, issue the animation command to return to frame #0. In this object, frames 4 through 39 are never specified.

When you are all done specifying your animation frames, click on 'CLICK HERE WHEN FINISHED.' If you want to keep your new animation, say yes to the YES/NO box. Then you will have to type a name for your new animation. Now you have created an animated object. However, just like when you create a new piece of artwork with GCSPAINT, you still must import it before you can place the animated object in your levels. Do this just like importing any other object in a library. You select the LIBRARY IMPORT/MODIFY function, then ANIMATED OBJECTS, and then IMPORT. Once this is done, you can place your animation in your level using the 'FILM' icon on the left side of the screen.

Summary of Animation Commands

Unconditional Branch

This jump command is called 'unconditional branch' on the operation list. When the object gets to that frame in its animation, it jumps to a different place. When you select 'UNCONDITIONAL BRANCH' from the menu, you are prompted for a frame number to branch to. In the flame example, you would put a zero in there, because you want to return to frame #0, the beginning.

Branch if Shot

This is fairly self-explanatory. This command works just like the unconditional branch as described above, except that the jump will only occur if the player has shot the animated object since the last time this command was encountered.

Invisible Blast

This just damages the player if he is close to the animated object. The closer to the animated object, the more damage he will receive. The radius of harm is about 700 cm. This blast will also kill enemy characters, which can make for some pretty neat situations in the game play. You could use this command to lure your enemies into a trap, for example.

Remove this Animation

This causes your animated object to disappear from the game play, never to return. For example, an exploding-barrel-animated-object would want to start as a barrel, then keep cycling through the first few frames (all with the same barrel picture). In one of those first few frames you have branch when shot, so that your object can jump out of the loop. Then it should display some explosion pictures, and trigger one of the above Invisible Blast operations. Finally, after the last explosion picture frame, you will want the whole object to disappear forever. This is when you would use this command.

Add a value to a Register See the section on universe registers.

Branch if the Player Close

Here, the object only jumps to the new frame if the player is within 600 or so centimeters of the object. If the player is not close, the branch would be ignored.

Warp to a New Level

This makes the player switch levels. Most normal level switches are done with platform objects, since platforms are fairly simple objects. However, if you want to transport the player somewhere else because of shooting something, this command is a good way to do it. You must know the 'Warp-to' number of where you want to go to. If you haven't yet specified the entry points to your various levels, then don't try to use this command.

Damage Player

This obviously causes damage to the player. The difference between this command and the Invisible Blast, is that this Damage Player command will hurt the player even if he is on the other side of the level. He doesn't have to be close to the animated object to get hurt. Also, this command will not affect any enemy units.

Play Sound Number

Look at the sounds.txt text file in your project directory. In the file there will be a list of sounds, and their reference numbers. If you want your animated object to play that sound, then use this command.

Skip Next if Reg != Val

This is a register command. If you don't know what a universe register is, then please see the chapter about universe registers. The exclamation point in front of the equals means 'NOT EQUAL'. There is another command for skip if reg = value.

Branch if no Guards

Like the other conditional branches, this one jumps to the specified frame if the player has killed all the enemy characters. This refers ONLY to the enemies created by clicking on the 'ENEMIES ICON'. It doesn't include bad guys that are really just animated objects.

Skip if Player Close

This causes the animated object to skip the next frame if the player is close. This is different from 'BRANCH IF PLAYER CLOSE' because you can specify the distance that triggers this command. For example, you can set this command to skip the next frame if the player is 1410 cm away. The distance is flexible.

Set Invincibility Use this to make the player invincible for a limited amount of time.

Assign a value to a reg Specify a value to be stored in a universe register.

Skip Next if reg=val See 'skip next if reg!=val' above. But notice the difference between '!=' and '='.

Display a Text Message

Read the chapter on text messages for more information about writing and compiling text messages. Once you have gotten your messages into the pstr.str file, you can have those messages pop up on the screen with this animation command. The messages

are referenced by number. This is a good way to make friendly characters talk at the player.

Modify a lighting register

There are several registers that control the lighting effects of the levels. The only way to change them from the default values is to create an animated object that uses this command. These are NOT universe registers. Lighting register #0 is in no way related to universe register #0, or any other universe register. This animated object command is the only way to modify the lighting registers. See the chapter on lighting effects for more information about these registers.

An example of a typical animated object

Now that we have covered all the animated object commands, let's get back to discussing the general methods of creating animated objects.

To make it more concrete, here is an example. Say you have two images of a guy waving his hand, guyhand1.vgr, and guyhand2.vgr. You put both these .vgr files in your library directory and import them so they show up on the list of symmetrical objects when you click on the TREE ICON.

Create a .vgr image of the guy standing there, and call it guystand.vgr, and import it also. Then you go back to the library, and choose 'animated object', and then 'design/create'. You say no to the 'set position' question, and then set up the animated object with the following frames:

FRAME #0 guyhand1.vgr no-operation (delay 2)
FRAME #1 guyhand2.vgr skip if player close (maybe 400 units)
FRAME #2 guyhand2.vgr unconditional branch to frame #0
FRAME #3 guystand.vgr no-operation (delay 6)
FRAME #4 guystand.vgr display text message #3
FRAME #5 guystand.vgr unconditional branch to frame #5

This guy will stand there waving until the player gets close. Then he will put his arm down for a moment and put up the text box. It is up to you to type in, and use PSTR.EXE to compile the text message before you try this out. If you forget to define the text message, you will get a critical error. But then you realize that shooting this guy doesn't kill him. That is easy to change. You make a series of death pictures of your guy falling to the ground. Maybe call them guydie1.vgr, guydie2.vgr, etc. Then you define them all in the object library as before. Then modify your animated object to be:

FRAME #0 guyhand1.vgr branch if shot to frame #7
FRAME #1 guyhand2.vgr skip if player close (400 units)
FRAME #2 guyhand2.vgr unconditional branch to frame #0

FRAME #3 guystand.vgr branch if shot to frame #7
FRAME #4 guystand.vgr display text message #3
FRAME #5 guystand.vgr branch if shot to frame #7
FRAME #6 guystand.vgr unconditional branch to frame #5
FRAME #7 guydie1.vgr no-operation delay 2
FRAME #8 guydie2.vgr no-operation delay 2
FRAME #9 guydie3.vgr no-operation delay 2
FRAME #10 guydie4.vgr unconditional branch to frame #10

This may seem a bit confusing, but if you go through the frames one by one, doing the frame commands, you will see how it works. Give it a try, and you will find that you can kill the guy either before or after you talk with him.

If you want, you can replace frame # 7's command to be one that sets a universe register. The object gets to frame #7 only if the player killed the standing guy. Thus you could have other animated objects or events triggered by this action of the player. You could have the game end, or have walls disappear that concealed bad guys. Or you could even have other animated objects that were just standing there suddenly attack. They would be sitting in a frame loop waiting for that universe register.

More info about Animated Objects

Animated objects are active objects, because you can tell one frame to change other frames, or even other objects. Go through each frame, clicking on the line in the scroll box. You can set the animation to a new position, give it a new command, or tell it to become another object entirely!

Usually animated objects don't move around. That is why you are asked if you want to make a stationary object when you start defining your animation. Creating moving animations adds to their complexity, so it is nice to be able to make an ordinary animated thing without getting into reference points and other things.

Your object could be a stationary, friendly character waiting to give the player a message. Or, your object could be a switch that is up on a wall, or perhaps a propeller fan in an air duct. In fact, making a fan would be easy. No special command or placement is needed. You would just make a few pictures of a fan blade at various rotations, and use the object library ADD/IMPORT function (called up from the EDIT menu) to define them as windowed wall type objects. Then you would put the frames in as animated objects. When you place one these 3D animated objects in your 3D world, the fan blade looks like it is turning.

But in other cases, you might want a moving object in the 3D world. For example, in a warehouse, you might want a rat to scurry out from underneath a crate, and then back. You make a few animation frames of a rat walking, and import them as symmetrical objects. Then you would set the frames to your newly added rat walking symmetrical objects. As you set each animation frame to be a new graphic picture, you would click on 'set position' to actually choose a position in your 3D world.

Here is a point which can be confusing. Although you are choosing points in your 3D world, just creating this animated object will not add it to your current level as a new object. Creating an animated object will not place it, even if it seems like you are placing it when you are placing the frame positions. This might seem a bit strange, but it is really useful to make objects which you can duplicate and reuse. You would hate to have to define a new animated object for each torch in your hallway, for example. In our warehouse case, you might want to have 16 rats, and 16 crates that they run out from. So you define the rat animation once and put it in your library. Now it is ready for use in any of your levels, or even in other projects that use the same color palette.

The fact that the animated object is never restricted to stay in the same place in your 3D layout, means that you must specify a reference point. The reference point indicates where you want to anchor your animation's movements. By default, the point of the first animation frame (frame 0) is the reference point. So when you place this animated object, the point where you click will be the point where the object specified in the first animation frame will appear. So in the warehouse example, you would set the positions for the rat object based on the rat coming out of a crate. You might want to start with the rat in the center of the crate in frame 0. Then you make the rat move about and return to the initial position. Then when it is time to select the reference point for the animation, you can leave it defaulting to the first animation frame. Or, you can click on the 'set reference point' button that appears when you save the animation object. Then just remember that when you place your animation in your levels that you want to place the rats in the center of a crate.

At the end of creating your animated object, you are asked if you want to save it. If you messed it up, and just want to start over, press the no button. If you want to keep it, then press the yes. If you press no, everything you just did will is lost. If you press yes, then your new animated object will be added to the current object library. In addition, a .fld file will be created that has all the animation information in it. This allows you to copy your animated objects from one object library to another.

How to use your animated object

Assuming you have created your animated object and put it in your object library, it is very simple to place it in your levels. Just click on the 'FILM' icon on your screen. You will then get a list of animated objects that are in the currently selected object library. Choose one and then you will be in 'place' mode. One click will put one of these animated objects into your level. If you want to place another, just click again. When you don't want to put any more of these objects in your level, just right-button click on your mouse. !!! NOTE !!! If you animation is not in the scroll box list, then perhaps you forgot to import it after you finished creating it. If so, reread the part of this chapter about the steps involved in creating a new animation.

If you want your animated object to be raised off the ground, you can select it, and use the SET-ELEVATION ICON just like you do for a wall or symmetrical object. Be careful when setting attributes on animated objects, however. Don't try to set 'DON'T DRAW BACKSIDES' if you have symmetrical pictures in your animated object.

What to do if your object causes Critical Errors

Animated objects are very complicated. It is very possible to make mistakes which make the 3D engine fail. Examples of bad things you can do are described below.

Jump to an animation frame that you haven't filled in
Play a sound that isn't defined in your sounds.txt file
Try to display a text message that hasn't been defined in pstr.txt and compiled into pstr.str
Forget to end your animation with an endless cycle or 'remove' command.

It is also possible to make mistakes when entering a reference point, thus taking the object out of the known universe. If you suspect that your animation object is causing an error when you try to run your level, try taking it out with the erase tool. Remember that your animated object will still be in the library even if you take all of them out of your level. You won't have to redo it.

It is very difficult to modify existing animated objects that move, and we recommend against attempting it. If you really must edit the positions of you animation frames, you must start with frame #0, and redo all of them.

MAKING SOUND WITH PIE 3D GCS

Importing your .WAV sound files

When you create a project, the basic sounds that are necessary for games are copied to your project directory. These include gun sounds, human 'HEY!' sounds and death sounds. There are two ways to go about changing these. One is to replace those .wav files with new .wav files of the same name. This is fine for changing the default sounds, but it can lead to confusion when different directories have the same filenames but different sounds.

When you come up with new sounds for animated objects to use, then you are you must tell the game engine to load it into EMS/XMS memory at start time. That way when your animated balloon pops, the boom noise is all ready to play.

Edit the text file 'sounds.txt' that is located in your project directory. The comments at the beginning of the text file will give info about the sounds that need to be defined for the game engine to run correctly. The format of the text file is very straightforward. There are three columns on each active line. An active line is a line without a ';' as the first word. The lines with the ';' as the first words are ignored by the game engine.

The first column should be a sound number. You can set your sound number to any number. But, it might be a good idea to assign a new sound a number greater than 31, since some lower sound numbers are already reserved.

The second column is sound priority. Zero is the highest priority, and any number higher is lower. If you make all your sounds priority 5, this should work fine. The Pie 3D Game Creation system can handle up to four simultaneous sound effects, so even at

low priority, a sound will rarely be superseded by another sound effect. You may choose higher priorities for you sounds, but if you go all the way up to zero, there is a chance of the new sound superseding the player's weapon.

The third column is the sound file name. The sound files MUST be EIGHT BIT MONO .WAV files. Sixteen bit .wav files will not work, and stereo .wav files will not work. The .wav files must be recorded at the rate of 17,100 kHz. If the sample rate isn't exactly right, the sound will seem speeded up or slowed down. Also, it is important to note that all the .WAV files must be smaller than 65535 bytes, or else they will fail. This is about 3.5 seconds of sound when recording at the correct speed of 17,100 kHz. If you want to verify the size of your .wav file, just look at the directory listing. If the file size is 65535 or under, the sound file is ok.

So now that you can define sounds, it is easy to get events in your game to use them. You can redefine the weapons' sounds (or door sounds, or the player gets hit sound etc....) just by overwriting with files of the same name, or by adding your new .wav file to the project directory and changing sounds.txt to use your new filename. You add new sounds to animated objects or other events just by adding the .wav file to the project directory, and editing the sounds.txt file. Just add more lines at the end of the sounds.txt file to define new sounds for animated objects. Then when you are creating animated objects, you can use the play sound animation command to play the new sounds.

Often, you want a sound to play every once and a while to set the mood. For example, in one of our games, we wanted a radio operator's voice in the background. So we added a radio talking .wav file, and put it into sounds.txt at the end. Then, we can make an animated object which doesn't move or do anything except to sit there and play this sound occasionally. In those cases, you usually want to place the animated object in some dead space (in part of the universe between walls that the player will never see.)

The farther the player is from the animated object, the quieter the sound will be. So if want to have sounds that can be heard no matter where the player is in the level, you will need to play a few of the animated objects around that level.

Making Music with the Pie 3D GCS

The Pie 3D GCS allows you to add your own general MIDI files to play music during the game play. Simply put your midi files in your library directory, which is in your palette directory. Specify the midi files using the musical note icon on the main GCS screen.

The library directory is the directory where all the .VGR image files for your level exist. For example, in the DEMO project that comes with the GCS, all the wall artwork was selected from the library called BASICLIB. If you have installed the GCS into the standard c:\p3dgcs directory, then that library directory is:

c:\p3dgcs\$rp9a\basiclib\

Load up the DEMO world LEVEL1, and click on the musical note icon. You will see your midi file listed there, if you have copied your midi file into the directory listed above.

Problems with midi music files

Unfortunately, the general midi standard is quite loose. Unless a piece of music is tailored to use a particular playback system, it will generally not sound that great. The two pieces of music included with the DEMO were tailor-made for the midi system used in the GCS.

When dealing with Sound Blaster/Adlib midi music, there are lots of limitations. You may only have so many simultaneous voices, and the drums are very fixed in nature. In short, most of the more complicated general midi stuff you might pull off networks or bulletin boards will need fixing before they will sound nice.

Another thing about midi files is that they must be less than 32767 bytes in size. Look at the directory listing of your midi file, and make sure it is smaller than this. If not, you will get a Critical Error #95 message when you try to test the game.

If one level in your game has music, then they all must have music. If you put music in some levels, but not others, then you will get a critical error.

You can have sound effects with no music, but you cannot turn off the sound effects and leave the music on. After making final, you use 'S1 T' for sound effects and music, and 'S1' for just sound effects.

If you put your midi files in your library directory, and the GCS is not putting your .mid files on the list of available music files, the most common reason is that you put the .mid files in the wrong directory. Or, you may be using more than one object library, and the current library is not the one that you put the music into.

INVENTORY ITEMS

The Pie 3D game engine can have objects that the player can pick up, carry around, and put back down. These are called inventory objects. Inventory objects are small symmetrical objects that lay on the floor. When the player approaches the inventory objects, an icon appears in a little area on the game play screen. This indicates that an object is close enough to be pickupable. The player can pick up an object with the 'I' key. When he does so, the inventory object is removed from the 3D world, and appears in the inventory area at the bottom of the screen.

Objects like keys just sit there in the inventory. When the player comes to a door, the inventory is searched by the game engine for the proper key. If the right key is there among the inventory items, the door opens.

Weapon objects make weapon animation files available to the player. Ammo items load the weapon. If the proper weapon isn't there, the ammo goes into the inventory until the player finally gets the right weapon.

The player can drop inventory items with the 's' key. When the user hits this key, a line appears under the first inventory item box on the bottom of the play screen. Then the arrow keys will move the line from box to box in the inventory area. When the player reaches the inventory item he wants to drop, hitting a return will move the object from the inventory into the 3D world.

It is very easy to use the standard inventory items that come with the game. Simply click on the bag icon to place an inventory item. Once you click on that, you will be looking at a scroll box list of inventory objects to place.

The list includes weapons, ammo packs, health packs, radar units, and many other things. It may be desirable for the user to change the appearance of these items. If you have several inventory directories, you can choose among them using the menu selection 'CHOOSE INVENTORY'. Selecting this will give you a scroll box list of inventory sets to use.

Complete list of the default placeable objects

Goo Gun This is a non-lethal weapon that immobilizes enemies permanently-- weapon inventory number 26.

Shotgun This is a short range weapon with devastating damage potential-- weapon inventory number 16.

Machine Gun This is a long range weapon-- weapon inventory number 15.

Hand Grenade This is a weapon that can be thrown-- weapon inventory number 1.

Rocket Launcher The player needs to pick up rockets after picking up the launcher-- weapon inventory number 25.

Generic Ammo This ammo contains some ammo for the machine gun. To make it contain another kind of ammo, you can customize it with the "EDIT ATTRIBUTES" command.

Shotgun & Machine Gun Ammo Adjust the amount with EDIT ATTRIBUTES.

Rocket Ammo By default, this contains just one round for the launcher weapon.

Gas Grenades This is a non-lethal gas that stays around after detonation-- weapon inv. number 28.

Bomb Pickup and plant the bomb, then hit 'b' key to make the bomb explode

First Aid Pick it up to recover from damage. If healthy players pick it up, it goes into inventory for later.

Key Unlock doors. See the chapter on creating doors.

Document This triggers a text box when it is picked up. Set "text msg. #" with "EDIT ATTRIBUTE." See the chapter on text messages for info about how to set the text.

Armor This protects the player from hits. Protection is adjustable with the "EDIT ATTR"command. Newdamage=olddamage/(1+armorvalue)

Gas Mask This device lets players walk through gas grenade clouds, unaffected.

Radar Pack Radar stays inactive until player picks up one of these.

The GCS only has one basic inventory object set. So, you may want to make your own custom set. This is as simple as bringing up GCSPAINT, and going to the inventory directory to change the picture used by the game engine.

Customizing appearance of inventory items

Like the enemies and object libraries, the inventory directory is inside the main palette directory on your hard drive. Since inventory items are so closely linked with your weapons, both the weapons and the inventory items are kept in the same directory. Since you might have lots of projects you are working on, you might have a few different weapon/inventory directories, each with different sets of pictures for your various projects. Inventory items, like any other 256 color images, have a palette or basic set of colors. That is why all the directories of artwork in the GCS are organized in palette directory trees. All artwork that uses the same set of colors is kept in the same place.

Say you installed the GCS into the directory c:\p3dgcs. Also, let's say you chose to use the palette called $rp9a. So all the artwork, weapons, inventory items, and enemy files that you use must be artwork that uses the palette called $rp9a. Each palette has a master directory. If the palette is named $rp9a, there is a directory that is in the c:\p3dgcs directory called:

$rp9a\ or the full path name is... c:\p3dgcs\$rp9a\

All the libraries, weapons & inventory items, and enemy artwork files that you can use in your game must be taken from this $rp9a directory. When you started your project, you told it to use the palette named $rp9a.

But to keep all those libraries (weapon files & inventory item artwork, and enemy images) organized, they all are kept in separate subdirectories within the master palette directory. So the inventory items you have chosen for your game, might be in a directory called dweapon\. Now this dweapon\ directory must be located within your palette directory $rp9a\, which is located in your GCS main directory.

So if you wanted to edit the basic inventory items, you bring up GCSPAINT. Load a .vgr image. Then click on the '..' double-dot icon to move to the main palette directory. From there, click on the inventory name in red. This makes you enter the invent\ inventory directory. There you can edit the .vgr files that make up the inventory items.

c:\p3dgcs\$rp9a\dweapon\

When it really comes time to make changes in the inventory items, we recommend that you use the 'CREATE DIRECTORY' menu choice from the FILE menu. Make a new weapon/inventory directory in your present master palette directory. This will copy the default set to your current master palette directory. Then you can edit these to your custom standards, and keep them there for any other projects that will use this palette.

Customizing the function of Inventory Items

As far as customizing the functions of the inventory items, you can do this to a limited extent. Many inventory items have adjustable parameters. For instance, health packs have adjustable amounts of healing, and ammo packs have adjustable amounts of ammo. Also note that these adjustable numbers are set after you place the item. Use Edit Attributes to set the number to the correct value. When you place the item with the bag icon, a default value is put down.

There are also other inventory items that use an adjustable parameter. The memo inventory items take a number for a text message. Then when the user picks up the item, the number value is used to reference a message that gets put up on the screen. For more information about how to make text messages, see the chapter in the manual on text messages.

To actually adjust these values, you place the item on your layout using the bag icon. Select the object with the selection tool, so that the object has a little red x on it. Make sure that just the one object you are interested in changing is selected. If more than one object is selected, this will not work. So after you have selected your new inventory object, you must activate the 'edit attributes' command from the EDIT menu.

When you pull up the EDIT ATTRIBUTES menu, there will be 4 red text boxes with numbers in them along the bottom of the menu. The first is a number which tells the engine what table entry to use for the icons and the function of the item. There isn't much use in changing this number with the attribute edit function. This is true unless you want to make a fake object that looks like one thing, but actually turns out to be something else when you pick it up.

The second number is the more important one. For health packs, the number holds how much healing that the health pack will do. Perfect health is 320. The default amount of health fixing is 80, but you can adjust it for each health pack you place in the 3D universe.

This can also be the text message number for memos. To make your text document for the player to read work, you first make a text message using PSTR.TXT (see the chapter on TEXT for more info), and then go to the "EDIT ATTRIBUTES" menu item

after placing your memo type object. There, you want to change the second red box to be the message number of the text you want displayed with the user picks up that item.

For armor, the second red box contains the armor value.

For ammo, the second number contains both the amount and the weapon type. How can you put two numbers in one number box? You take the weapon inventory number, multiply it times 256, and then add number of bullets. For example, for machine gun ammo, you take the machine gun inventory number, 15, and multiply by 256 to get 3840. Now if you want the ammo pack to have 41 bullets in it, add 41 to 3840, and you get 3881. So put 3881 in the second "EDIT ATTRIBUTE TEXT BOX." Now your ammo pack contains 41 machine gun bullets. This is how you can use the `Generic Ammo Pack` inventory item to contain whatever weapon ammo you want.

Grenades and rocket shells are like ammo too. You can actually make grenades and shells that contain more than 1, but that might be confusing. It might be desirable to change the appearance to that of a crate of shells or something. Then set the edit attribute's second number to make it contain multiple ammo shells.

Making your own special inventory items

Since the action taken when you pick up or put down an inventory item is controlled by the game engine, the basic set of possible items is fixed. However, it is possible to set universe registers when certain items are picked up. In response, some animated objects could change their behavior accordingly. However, the process is rather involved.

To start with, ANY symmetrical object can be made to be an inventory object. Strangely enough, all that you have to do is select the object, and then set the 'DON'T DRAW BACKS' attribute with the EDIT ATTRIBUTES command. However, there is more to it. You need to tell the engine what the inventory item does, and what icons to use for the pictures on the game play screen. This would have to be done with the 'edit attributes' command.

To see this, place an ammo pack with the normal inventory 'bag' icon. If you have any other objects besides the one ammo pack selected, then the red number boxes won't appear when you bring up the EDIT ATTRIBUTES command.

When you bring up the EDIT COMMAND box, look at the first text box. See the inventory number of the ammo pack. Every inventory object has a type number. This is how the engine knows which icon to put up when you go near an object. The engine keeps a table in memory with inventory numbers and object numbers. These object numbers give the engine information about which icon to put up and what object to create in the 3D world when the player drops an object.

Any inventory object that you make by setting the EDIT ATTRIBUTES command 'DON'T DRAW BACKS' will act like a hand grenade if you put a 1 in the first number box. This is because the inventory number for the hand grenade object is a 1. If you look in the earlier list of objects, you will see that the goo gun is listed as weapon inventory number 26. If you set the attribute

'DON'T DRAW BACKS' with a '26' in the first red box as your are editing the attributes of a symmetrical object, then no matter what the object looks like, it will be a googun when the player picks it up. However, once the user picks up the object, the original identity of the object is lost, and when he drops it again, it will look like whatever it is supposed to be. That is because what it looks like when dropped is information pulled from this inventory table mentioned earlier.

Figuring out which species 2 object corresponds to the number used in this table is available in a text file.

The other thing in the table is a number which tells what the object actually does. There are 15 different runcode numbers. Runcode 0 is the basic inventory type. You pick it up, you put it down, it doesn't do anything at all. Runcode 1, is a bomb object. Press the 'b' key, and this inventory object explodes, killing any enemies who might be near it. The other runcodes will be covered later.

There are two major text files that control what pictures get loaded into your 3D game for use and what icons and purposes the inventory objects have.

Weapdef.txt is a file that really contains all the object definitions for your weapons and inventory things. Most normal objects like walls and trees and animated objects are defined when you use the library import/modify command in the GCS. However, the inventory objects remain constant in every level of your game, so they are fixed and defined in this text file instead.

Copies of the weapdef.txt file are located in many places in the GCS directory tree. The original weapdef.txt is in the 'weapons' directory off your main p3d directory. Do not change this one, as that is the one used when creating a new weapons directory. First use the command 'CREATE NEW WEAPONS DIRECTORY' to make the directory, then go and modify the files in the directory you just created. See the section above on changing the appearances of the inventory objects.

Additional instructions about the file format and modification procedures are contained in the weapdef.txt file itself. The first hundred lines of so of that text file are comments. The tables are later at the end of the file.

The basic point is that the weapdef.txt file is a table. Each line defines another symmetrical object that is available in your level. In each line, you begin with a '2'. This tells the 3D engine that the object will be a symmetrical one. The next number is one you come up with. This is called a handle. A handle is just a numerical name for an object, somewhat like a Federal Express tracking number.

The next item on each line is the image file name. This must be a .VGR file created by GCSPAINT. Then comes the horizontal 3D size in centimeters, followed by the vertical size. Once you have typed your object line in this file, this object can be used as an inventory object in the GCS. Placing these objects cannot be directly done, but we will get around to that later.

For each inventory item, there are two necessary objects. One is the object icon that appears, the other is the object that gets created when you drop one of these items. So to create a new inventory item, you must first define two symmetrical objects in

the WEAPDEF.TXT file. Be careful not to use a handle number that is already in use by another object, either in Weapdef.txt, or in the GARDEF.TXT in the enemies directory.

The object that makes the icon object MUST be 16x16 PIXELS. Check the image size with GCSPAINT. The world size doesn't matter. The object that the player drops can be any size.

So you have defined your two objects, now how do you associate them with a pickup object that has a function? To get your handle numbers for the icon into the inventory table, you need to modify the pickups.txt file.

This file is just a table of inventory items. The first column is the pickup number. Each type of item has a number. The googun weapon is 26. The machine gun is 15. The hand grenades are inventory type 1. These are not handles, and they are not runcodes. These are inventory type numbers. You can make up your own inventory numbers, if they are smaller than 64. It is up to you to make sure you don't pick one that is already in the pickups.txt list.

The next two columns are the handle numbers of the 3D objects and their icons. This is how the game engine knows what icon to put up when you get near an object. It looks at the inventory type number that is set in the first 'red box' attribute of the object. It uses that inventory type number to look up the line in the table. It then sees the handle number of the icon, and then can put up the icon picture. For example, if you place a goo gun, and look at the first red box in the 'edit attributes' command, there is a 26 there. This number corresponds to the weapon inventory number described earlier. Now look in pickups.txt for the line that starts with a 26. This line in the pickups.txt file defines the goo gun. If you look at the two numbers after the 26 in the pickups.txt file, you will see some numbers.

```
p 26 2130 2120 7
```

What are 2130 and 2120? To find the answer, go back to WEAPDEF.TXT. You will see that 2120 is handle number for the goo gun icon, and 2130 is the handle number for the object that gets created when you put a goo gun down.

```
2 2120  goosmall.vga     64     64     ;NEW GOO GUN  icon
2 2130  googun.vga       180    40     ;NEW GOO GUN on ground  (z=180)
```

So what is the '7' in the last column of the pickups.txt entry for the goo gun? It is the runcode! That is how the 3D engine knows what to make this inventory object do. You'll find that runcode 7 is the weapon runcode. So this is how you make your inventory object. You make your objects .VGR files for icon and drop object. Then you edit WEAPDEF.TXT and assign handle numbers to the objects with vertical and horizontal 3D world sizes. Make them small, maybe 50 cm by 50 cm. Make sure you don't use handles that are already in use, or you will get critical error #87. Don't assign handles smaller than 2000, or you may collide with some wall section handles.

Then you edit pickups.txt to add your new inventory type line. Choose an inventory number. Make sure your new number is

not already in use in the pickups.txt file. Then put in your handle numbers in WEAPDEF.TXT. Make sure the .VGR image you are specifying is 16 pixels by 16 pixels in GCSPAINT. If it isn't, you will get Critical error 167 and 168. Pick a runcode and put in the last column of your new entry in your pickups.txt file. We will give a list of runcodes and their functions later. This is an example of a definition table for the pickups.txt:

```
p TYPE 3D#  ICON#  RUNTIME ;COMMENT
                    CODE
-------------------------------------------------------
p 1   2050  2050  0     ; grenade is always pickup type 1
p 2   2151  2151  0     ; key is pickup type 2
p 10  2980  2980  0     ; explosion
p 11  2169  2169  0     ; kick me down
p 12  2157  2167  3     ; regular memo
p 13  2150  2150  0     ; gold key
p 14  2158  2158  0     ; purple key
```

Now you are almost done, except for one last hairy detail. You can't place items defined in the WEAPDEF.TXT directly. The GCS only allows you to place items defined in the object libraries. By putting your object into the WEAPDEF.TXT file, you have defined it for the game engine, but the GCS layout editor isn't smart enough to know about objects in the WEAPDEF.TXT file. So how do you place your new item?

You must define the object again, but this time in the ordinary manner using the object library feature of the GCS. Give it the same size as you gave it in weapdef.txt. No need to define the icon again, only the 3D world object. Once you have put it in the GCS object library, then you place it like any other symmetrical object. Then you select it and click on edit attributes. You click on 'DON'T DRAW BACKS', and then you set up the first red box to your inventory number. If your object takes a value like a health pack or ammo does, then put that number in the second red box. What this second red box value does is totally dependent on the runcode. Remember that the runcode is in the table, so you don't set that on each object that you place. When you put the inventory number in the first red box, the engine will pick up the runcode from the pickups.txt table. You are now finished.

So what are the various runcodes and their functions?

0 baseinvfunct: No special functions, just pickup and drop like keys

1 bombinvfunct: This pickup type explodes with the 'b' key

2 mineinvfunct: This pickup type is normal except it kills careless guards who come too close. The object will explode.

3 memoinvfunct: This displays text pointed to by 'value' when picked up.

4 bmapinvfunct: It shows an image indicated by the 'value' when picked up. This is kind of subtle. You need to have pstr.txt string number devoted to just a .VGA disk filename. The number you put in for this object will reference that string. The engine will look up that name, then load that image file and display it on the screen.

A new feature has been added tobmapinvfunct. If the value given in the pickup object is less than 256, then the above paragraph is accurate. However, if the value is over 255, then the top byte of the value is taken as a universe register number. The actual string used to pick up the filename then has the value of that universe register concatenated onto it.

5 ammoinvfunct: If the gun needs ammo, it is automatically loaded. Otherwise, the ammo is added to the inventory.

6 faidinvfunct: Applies health to the player. It is specified by value 319 = max

7 weapinvfunct This is a weapon with an ammo amount in 'value'.

8 reginvfunct: This adds a value to a univ. reg: ureg[val>>8] += val & 0xff. To make an inventory object that sets univ. register 10 to 1 when picked up, place one of these objects and set its value to 10*256 + 1 = 2561 Then when the user picks this object up, universe register 10 will be incremented. When he puts the object back down, the value will decrease by one again.

9 bigbombinvfunct: No longer used.

10 basinvfunct: No special function

11 basinvfunct: No special function

12 radarinvfunct(): Pick this up for radar to work

13 gasmaskinvfunct(): Pick this up to stop the effects of the gas

14 jetpackinvfunct(): Does nothing, but modifies a universe reg. when picked the object is picked up

15 armorinvfunct(): This cuts hits by a factor of three when wearing armor

So how do you make a new inventory object and place it in your level? Say we wanted to make a magic amulet that will open a secret door when the user picks it up.

1. Go to the GCS and open your level. Click on "MAKE NEW WEAPONS DIRECTORY." Call it myweap.

2. Draw a 16x16 picture of the amulet for the icon. Save it in GCSPAINT as amuletsm.vgr. Save it in your myweap\ directory that is in your palette directory (probably $rp9a)

3. Draw a picture of the amulet for the 3D world. Save it in GCSPAINT as amuletsm.vgr. Save that in your myweap directory too.

4. Edit weapdef.txt in your myweap directory. Make a new line for each of your new .vgr files. The size for your icon in the 3D world doesn't matter because the icon .vgr never goes in the 3D world, but put a reasonable size in anyway. Make sure you pick handles that are bigger than 2000, and handles that have unique numbers that aren't used elsewhere in weapdef.txt or gardef.txt in your enemy directory. If a handle number isn't unique, you will get a critical error 87 when you test your level. So you choose the following to add to your weapdef.txt: 2 2950 amuletsm.vga 50 50 2 2951 amulet.vga 50 50

5. Edit the pickups.txt file in your myweap directory. Put a new line in with a new unused inventory number that is less than 64. Put in the handle numbers for your objects, and use runcode 8, which is the kind that increments a universe registers when picked up. Here we chose an inventory object type number to be 40. p 40 2951 2950 8

6. Now start the GCS, and make sure every level of your project uses the same weapon directory, namely myweap. You can't have different levels use different weapons directories, because you can carry inventory items from level to level in the final game. If you try to mix weapons sets between levels, within the same game, you get critical errors.

7. Now define amulet.vgr with the object library IMPORT/MODIFY option. Import it as a symmetrical object.

8. Place one in your universe, then select it and click on the object attributes command from the objects pull down menu. Set the push-button called 'don't draw backs'. Then set the number in the first red box to 40. Now the second box is the one that changes depending on the runcode. We gave our new inventory item a runcode of 8 in the pickups.txt. If we want this placed object to add one to universe register 20 when we pick it up, then we put 20*256+1 in the second red box. This comes to 5121.

Now we are done. The player sees an amulet on the floor. When he approaches, the amulet icon appears in the pickup window. He hits the 'I' key to pick up the amulet. When he does so, universe register 12 is incremented. If you had an animated object that was a wall that sits in a loop waiting for universe register 12 to be non-zero, then that wall could disappear.

This process may seem daunting at first, because of all the numbers to keep straight. But eventually it becomes easy, and with these universe register items, can make for some pretty interesting puzzles.

ARTWORK FORMATS AND ISSUES

There are three basic types of objects in the Pie game engine. Solid textured walls form the bulk of the wall sections. Windowed

or shaped walls are necessary too, as are symmetrical or stand-up items. This chapter will go into some detail about these objects.

What is an image file?

All textured walls start as picture files. A picture file might be a .gif, .pcx, .bmp, .vga, or .vgr file. All these types of data files are just flat pictures. For years and years, image files of these formats have been traded, swapped, and sold. All national computer networks like Compuserve, America Online, and Genie have lots of these image files for people to download and look at. Typically you need a viewer program that can read these disk files, and display the picture on your VGA screen.

If you have downloaded pictures and looked at them, you will notice that they come in all different sizes. The smaller files are usually 320x200, while some huge ones are 800x600 or larger. So when we say 320x200, what units of measure are we using? Certainly not inches!

We are talking about pixels. A pixel is the smallest individual singled color spot you can display on your computer. Your graphics screen is always made up of thousands of little colored rectangles. The computer can set each of those little blocks on your screen to any color. In the old days, the pixels were very evident. It was plain on the original IBM PC's that all the characters on the screen where made from dots. Now, with the big advances in graphics technology, it is getting harder to see the individual pixels.

What is resolution?

As much as the technology has changed, the fact that all the characters and pictures on your computer screen are made up of pixels has not. There more pixels on your screen than there used to be. Using the proper technological terminology, one would say that the resolution has improved. Resolution refers to the number of pixels per inch on your monitor screen. In printers, resolution is measured in DPI, or dots per inch. They don't use that measurement in graphics cards, because the DPI depends on your monitor size. In graphics screens, the number of pixels on the screen is given. In fact, instead of the actual total number of pixels on the screen, the resolution is given by the number of pixels from side to side, times the number from top to bottom.

So if you have a 320x200 graphics screen, the number of pixels equals 320 times 200 or 64,000 pixels. When you are playing DOOM, or Wolfenstein 3D, this is the number of pixels on the screen. By today's standards 320x200 is a pretty low resolution video mode, but it is still good for games, because the computer has a lot less work to do to keep the screen refreshed.

Why is resolution so important?

The picture file size is closely related to the resolution of the picture. An image with a low resolution of 10x10 pixels would be very limiting. You could never show a picture of a neighborhood street with shrubs and houses with doors. If you tried to reduce a photo like that to 10x10, all the pixels would look like a grid of blocks of differing colors. Maybe if you if you blurred your vision by squinting, your eye could make out a hint of some large features, but overall, the image would be useless.

You can expand a low-resolution image to the full screen. But if you do, you will see a very blocky image. Since you shrink or enlarge the image to any size, the measured size isn't really important in an image file. What is important is how many pixels are in it. The more pixels, the finer the details can be. Experts should stop skimming here, because this part is important.

So why are we going on and on about this? It is important for the game designer to be aware of image resolution. Whenever you think about trying to make a picture for your game, you have to be realistic about what you can do and what you can't.

For example, say a designer says: "I'm making a mystery game. The player goes into a room, and sees a TV. He has to notice what channel the selector was left on...." This would be impossible in a game like this unless the TV knobs were bigger than your head. The resolution just isn't there for details.

You usually want your wall sections to use artwork of about 64x64 pixels. This is the standard size used in Wolfenstein 3D and DOOM. Why not have more pixels in the wall sections? Well, with the GCS you can. However, there are tradeoffs. The foremost is RAM. Due to decisions by certain large companies, the DOS operating system was never allowed to modernize as computers became more powerful in the 80's. Therefore, we are stuck with same memory limits as the original IBM-PC. The 640K limit is what we are up against here.

With the game engine running, there is about 300k or so of RAM left for image data when playing the game. The game engine DOES use EMS or XMS. But, because of the way the game engine is designed, EMS or XMS memory can only be used to store .WAV sounds and weapon images, not wall patterns. This is because the engine uses a scan-line Z buffer. The upshot is that the walls are drawn in little bits. To use EMS or XMS for the wall data would mean yanking each wall in from EMS or XMS hundreds of times for each animation frame.

Memory Limitations and Image Resolution

Each pixel of 256-color artwork takes up one byte of RAM. A 64x64 picture for a wall section would take 4096 = 4k RAM. If you wanted to make a game with no weapons, inventory items, or enemies, you can probably have 75 different wall images in your levels. This is because seventy-five 4k wall panels take up 300k.

In real life, you want inventory items, and enemy characters, trees, and weapons. This takes up space. In the Pie 3D GCS, you usually have 10 or 20 unique wall sections in any particular level.

But let's say you wanted high-resolution artwork for you wall panels. The engine can't take any image that has a dimension bigger than 256. The largest image that could work is 200x200. This image contains 40,000 pixels, or ten times more than our 64x64. This means ten times fewer unique wall panels in your levels. In practice, we have never gone higher than 128x128. For walls that we want to blow up to bigger sizes, we may go up to 96x96. There is also no reason why the walls have to have the same dimensions horizontally as vertically. Often we make walls with resolutions of 64x104, especially when doing rooms with high ceilings.

So, be realistic about what you want to put on your walls. Walls with messages written in the artwork don't work very well unless the messages are of the four-letter single word variety. The problem is that even if you make a 128x128 wall section, the player can read the letters only when he is very close to the wall. That is fine, but then you realize that the player can't really bend over to read any writing that might be at knee level. You have wasted 32,000 bytes of precious RAM on a 20-character message. This really comes into play with using digitized artwork from a scanner or video camera. You will find that getting human faces to look right is a real challenge. Of course, if you are interested only in a fun demo, you can use a higher resolution image, since you don't need enemies or the other things.

Getting image files into the 3D world

In the GCS, the levels are made out of objects. Tees, enemies, ammo, and walls are all objects. In fact, the walls are built panel by panel, just like modular construction. Erecting another wall is nearly the same process as placing a tree or a health-pack. Almost every kind of object has an image file associated with it. In GCS, all images have to be either .VGR files, or .VGA files. If you want to use one of your .GIF, .PCX, or .BMP images, then load those pictures into GCSPAINT one by one, and save them in the .VGR format.

Images vs. Objects

The Pie 3D game engine stretches images onto rectangles that are in the 3D world. Imagine being in an empty room with white wall panels all around you, and your alien master gives you a paint set, and a photograph that you must paint on each wall. He tells you to paint the photos on the wall with no border or white showing through. This is what the game engine does. Each object has a horizontal and vertical size (that defines the size of our wall panel), and an image (the photo). The game engine stretches that picture in the 3D world so the image exactly covers the specified size. The alien gives you a normal rectangular photo for you to paint on a very tall and thin wall panel. Now you have to distort the picture to paint the wall without leaving anything unpainted. The game engine does the very same thing. If you assign a regular picture of a person to be painted on a very tall thin wall, you will have a sideways-compressed image of a person on that wall. It would look like a fun-house mirror reflection.

So you can't just tell the game engine to put an image into the game. You must tell the game engine how big to stretch the image. For a brick wall section this is fairly straightforward. Most typical walls are 400x400 units in size. Whoa! But didn't we just say that the largest image we ever used was 128x128?

Bitmap image resolution vs. 3D world size

Now we have to deal with the 3D virtual world. In there, the distances are measured in centimeters. Since everybody loves the metric system, this should come naturally, right? The unit of measure really doesn't have any impact on world design. The bottom line is that a wall panel that is 400 units long in the 3D world looks about 12 feet wide. Now this says nothing about the

resolution of the picture that the engine paints on this 12x12 wall panel. We could stretch a 32x32 pixel image on there (it would appear blocky, though), or we could put a 64x64 pixel image on it, or maybe even a 104x64.

So that is how the typical wall is 400x400 centimeters big in the 3D world without violating our 128x128 pixel size limit. The 128x128 pixel limit is for the image artwork only. The 3D virtual world size of the object is very different.

Typical resolution and world sizes

Hopefully we have drilled that distinction into the ground. Let's just add that the most typical wall panel is 400x400 cm in size, and has an image on it that is 64x64. Keep in mind that if you tell the game engine to stretch a tiny 8x8 pixel image up onto a huge 800x800cm wall, it will try. However, it is possible to cause a numerical overflow in the 3D calculations if you try to make do stretches or compressions that are too obscene. In general, stick to images with horizontal or vertical dimensions of about 32-104 pixels. Wall dimensions should be between 50 and 600 units wide. You will not encounter those overflows. You might also experience trouble when making very short wide walls, say 800x50. So, be careful when making steps or guardrails.

Foolproof dimensions? If you want everything to be easy, then always make your images 64x64 pixels, and the sizes of the walls 400x400 cm. If you want to make things really hard, make your walls have random dimensions like 477x613cm. Actually, this only makes it hard for you to match up the walls of adjacent rooms. The odd numbers do not affect the speed or smoothness of the game engine. It should be noted, however, that there is no reason to keep 'nice' pixel dimensions in your image files. An 87x71 pixel image is just as easy for the engine to stretch onto a wall panel as a 64x64 pixel one is. The 64x64 pixel standard is merely convenient.

Walls with shapes and holes

Walls that are not rectangular are no problem with the Pie 3D engine. In fact, walls with a few large cutouts should be no problem. Examples are walls with windows, doors with rounded tops, a prison cell wall made of bars, or fences with gaps you can see through. All these are as easy to do as solid walls.

Simply make the invisible parts black. If you want a brick wall with a big round hole in it, then load up your brick wall image in the paint program. Use the circle tool to paint a big solid black circle on the image of the wall. Now the trick is that the black you use must be the first color of the palette. In GCSPAINT, the first color is color 0. You can read the color numbers on the bottom of the screen when in GCSPAINT. Choose your black from the upper left corner of the palette grid.

When the game engine draws this kind of wall object, it stretches the image exactly like a solid textured wall. The only difference is that it never paints the color 0 (black). So whatever was behind the wall shows through.

So, why bother having two types of walls? Why not make them all shaped walls? If you want to make a solid wall, then just don't put any black in it, right? Well, shaped walls take more computer time to draw, because there is all that decision making

involved. It was much more efficient to have two kinds of walls. However, if you really want to, there would be no problem with defining ALL your walls as shaped ones. You might find the animation isn't as smooth during game play, however.

Limitations in the shapes

There is a limit to the little bits of black in your image. We call these black areas 'holes'. In general, any cutouts will be fine. However, certain patterns will cause critical errors. When the engine first looks at the image, it pulls your picture apart into vertical strips, one pixel wide. Then it looks for black sections of these strips. Each area of black pixels on a strip is called a 'hole'. You can have up to 23 or 24 holes appear on any single strip of your image. Also, there is a total amount for the whole picture that you cannot exceed.

Shapes & holes to avoid

If you want to know what images will be slow drawing and hard for the engine to deal with, then remember that the game engine processes these images in vertical strips. So the image with the highest number of holes would be an image of Venetian window blinds. As you go down the image from top to bottom along any column of pixels, you see image going from beige to black, beige to black. Each blind strip causes another hole. But now turn that Venetian blind image on its side. Now as you go down the vertical column of pixels, if you happen to start on a blind blade, there are no holes as you go down. If you start on a black pixel, then the whole thing from top to bottom is just one hole.

But don't be afraid to try different textures. If the engine doesn't like your texture, it will say so when you test your level. You will get CECODE 116, or CECODE 19, both of which are error messages relating to the number or nature of holes in your image. Most walls with holes are OK. Some examples of bad things to try are chain link fences, Venetian blinds, and images peppered with tiny holes. All these have too many holes to deal with. Don't be afraid to experiment. Critical error codes don't hurt anything.

Symmetrical Objects

Symmetrical objects are things like trees, floor lamps, and chairs. Unlike walls, which change appearance depending on whether you are looking at them face-on or edgewise, symmetrical objects always look the same. The viewing direction does not matter because they always rotate to face you. These stand-up objects are really just shaped walls that always turn to face the player.

So to make a tree, you draw a tree on a black background. You put the size at 400x600cm in the 3D world. The engine draws that wall facing the player. It doesn't draw any of the black background between the leaves and around the outside edge of the image. Voila! A tree appears in the 3D world! So the important thing to remember about these objects is that they are stretched out over a rectangle, just like the images on the regular solid textured walls. The panels that these images are stretched onto have

horizontal and vertical size in centimeters, just like the other walls.

Just like the shaped walls, the images you make are subject to the same hole restrictions. If you want to make your own shrub, don't try to have see-through areas between each leaf. You may have trouble. Most reasonable trees, chairs, and airplanes will be nowhere near this limit.

Ceiling and floor images

You may can make floors and ceilings, but there are some limitations. The whole floor is made from one repeating pattern. So when you choose the image for the floor, you are choosing it for the whole level. Each level can have its own floor and ceiling, however.

The ceiling and floor images MUST BE 64x64. This is because the program does special tricks to increase drawing speeds. The 3D world size, expressed in centimeters, is ignored with floors and ceilings. The floor tiles will always be the same size in the 3D world.

Because your image will be repeated endlessly on the floor, the patterns you put on your image will be discontinuous as you go from one panel to the next. Sometimes this effect is desired, like when you are making floor tiles that look like square tiles. However, when making a dirt floor, you would like the user not to be able to see the tile boundaries. This means blending the color on the top of the image with the pixels on the bottom, and blending the left with the right side. GCSPAINT can handle this for you in the f/x sub menu. You can smooth the edges of your image to improve their appearance in your 3D world.

VGA vs. VGR image files

Note that there are two types of image files that the 3D game engine can read: .vga files and .vgr files. Typically, all your walls and other objects are .vgr files, and all the big things, like the backdrop, are .vga files.

The file types are so close in format that they are interchangeable for the most part without a problem. For example if you used the DOS copy command to copy a 64x64 pixel wall.vgr file to wall.vga, you could still read it in as a .vga with GCSPAINT. However you would notice that the image was rotated 90 degrees on your screen.

This explains what a .vgr file is. It is just a rotated .vga image. The 3D game engine can read in artwork more efficiently if the images are rotated 90 degrees. So the .vgr file format has the pixel data rotated 90 degrees internally. You never know this when you bring up your images with GCSPAINT, because GCSPAINT puts them up on the screen right side up although the image is on its side in the .vgr file.

The only files that need to be rotated with the game engine are those that are in the 3D game play world. Pictures of the hitpoint guys, (HTPT0.VGA) and the game play screen (BACKDROP.VGA), the weapons identification 'fat' files

(GUNNONE.VGA) all are put on the screen as regular flat pictures, not in the 3D viewport. That is why they are .vga files.

The wall panel artwork, however, needs to be rotated 90 degrees. You could do this saving your artwork as a .vga file, but make sure that you rotate the picture onto its side before saving. Then rename the .vga file as a .vgr file, so that the GCS will let you import them.

There is really no reason to do this except to illustrate the difference between .vgr and .vga files. This is necessary because when using GCSPAINT, you must keep aware of which kind of file you want to load or save.

GCSPAINT is kind of strange in that it never asks you whether you want to save as .VGR or .VGA. If you call GCSPAINT from the icon in the GCS program, you will always be saving as .vgr, although you can read both .vgr and .vga files. If you quit the GCS, and run GCSPAINT from the DOS command line, then all images you save are written to .vga files. This is good, because some bigger files that you might like to edit, like BACKDROP.VGA, or BACKG40.VGA are too wide to be saved as .vgr files.

So to change these large .VGA image files, you need to exit the GCS and run GCSPAINT from the DOS command line.

GCSPAINT

You don't need to hire an artist to make great looking graphics for your 3D project. GCSPAINT is an image workshop containing all the tools and special effects you'll need to create original artwork or to manipulate existing images. Click on the GCSPAINT icon. You are now looking at the GCSPAINT screen. The main menu is located on the lower right side of the screen. The current palette of 256 colors is located on the lower portion of the screen.

GCSPAINT help

Click on the "?" icon for an excellent introduction to GCSPAINT's many features. Help screens explain how to use each tool and special effect as you click on the various buttons on the main menu.

Disk commands

Click on the main menu's disk icon. You are presented with several options. Click on the various items to test their functions.

Create New Image

Create a new image by dragging the size box to the desired dimensions. Please note the maximum image size in GCSPAINT is 192X150. If you want to work on larger images, exit the GCS and run GCSPAINT from the DOS command line. Now you can work on full screen 320X200 images.

Save Image

GCSPAINT saves your images in the .vgr format. Limit your file names to eight characters.

If you need to save your image in the .vga format, you will have to run GCSPAINT from the DOS prompt.

c:\p3dgcs\gcspaint.exe.

This enables you to save large images like title screens and background images. Conserve RAM memory at game time by keeping your images small. Typical walls should be 64X64 and typical characters should be about 60 pixels tall. You can scale down large images with the resize command. Try to maintain a reasonable balance between appearance and size when saving your images.

You should also avoid leaving a lot of unused black background in your images. Use the move command to put your images in the upper left corner of the screen. Then, use the resize command to crop the excess black background.

In addition, you should also test images that will later become shaped and symmetrical objects to make sure that they don't have too many holes. Color 0 is not drawn by the 3D game engine, thus those areas are see-through. An abundance of see-through holes can sometimes cause critical errors when you test your levels. An easy way of testing for this pitfall is to use the color replacement feature to change areas drawn with color 0 to a bright color. Then, you can clearly see how many invisible holes are in your image. Use color replacement again to refill those unintentional holes with a color other than color 0.

Merge Image

This command lets you combine two separate images. A problem arises when you try to import an image that is bigger than your current image. A warning will flash momentarily at the bottom of the screen and the operation will fail. Use the resize command, without scaling, to make your current image big enough to accommodate the imported image.

View as Tile

You will undoubtedly be working on wall, fence, or foliage artwork for your project. A common problem with artwork repeated or stacked in your 3D world is that distracting patterns appear. Edges look harsh and ill-fitting. This command gives you a preview of what your image will look like when it is placed side by side or stacked one on top of each other in your 3D world.

Hidden Main Menu

Large images sometimes cover up the main menu so you can't see it. To view the hidden main menu, move your mouse pointer to where the main menu normally appears and hit the space bar key. Once you select a command, the main menu disappears

again and your image fills the whole screen.

Palette

The GCS works with one palette of 256 colors. All the images that you use in your game must use this one palette. Each color has a number from 0 to 255.

The palette box displays 64 of the 256 colors on the lower portion of the screen. To see the other 192 colors, simply click on the arrows on the right side of the palette box.

Select a Color

Move your mouse pointer over the desired color in the palette box. Click the left mouse button to select a color. You can also pick up a color from your image. Move your mouse pointer to a pixel in your image. Click the right mouse button to select that color from your image.

Hidden Palette Box

Large images sometimes cover up the palette box so you can't see it. To view the hidden palette box, move your mouse pointer to where the palette box normally appears and hit the space bar key. Once you select a color, the palette disappears again and your image fills the whole screen.

Load a Palette

Images do not always come with their own palettes. These images appear in wild colors when you load them, because they are drawn using a default palette. Remember that the GCS uses only one palette of 256 colors at a time, so make sure that you are loading the same palette that will be used throughout your game.

Modify Palette

It is possible to customize your own palette. You can adjust each individual color by adjusting its red, green, and blue components. You can also work on several colors at once with the trend option.

Match Palettes

GCSPAINT gives you the option of matching the palettes of imported images to the palette currently in use. This is particularly helpful when you are using scanned images for your game. This feature enables you to make all kinds of artwork compatible with the palette that you are using for your game.

Save Palette

You should always save the palettes of images before you try to match them to the master palette of your game. If the palette matching operation doesn't turn out well, you will want to reload the image with its original palette and try again. If you don't save the original palette, the original image will be ruined forever and you won't have a second chance.

Draw Tools GCSPAINT equips you with many powerful drawing tools: lines, rectangles, circles, ellipses, and polygons.

Interpolation

This command creates a morphing effect between two polygons. You can specify the number of intermediary shapes in the morph. You can also set the number of colors involved in the morph. Be warned, however, that the operation fails if original polygons have too many sides.

Settings

You can set preferences for GCSPAINT. Choose this command to specify the brush size. You can even design brush patterns. In addition, you can set prompt delays to suit your needs. Memory settings are also adjustable.

Text

Two .chr files come with GCSPAINT, although you can put public domain .bgi files in the same directory as well. Select the size and style of the text you want to put in your image. If your text operation fails, it's probably because the text was too big for your image. Try again with smaller letters, a more compact font, or a larger image.

Color tools

GCSPAINT is loaded with first class color manipulation tools.

Fill You can fill areas of color with either solid colors or patterns.

Color Replacement

This tool lets you switch colors in your image. This is a big time saver since you don't have to add the new color to your image pixel by pixel.

Random Replacement

This handy tool lets you replace a percentage of a color in your image with a new color. This is especially useful when doing artwork that requires a haphazard pattern such as dirt floors or carpeting.

Color Phase This tool fills an area with a gradient. Pick a color and start the phase on that color.

Color Sunburst This is similar to the color phase tool, except that it makes gradient fills in rounded areas. Pick a color, then start the sunburst on that color.

Special effects This is the most powerful set of tools in GCSPAINT.

Outline

Why spend your time drawing borders around areas by hand? Select this tool and your border is done in a flash! Your outline can be done with or without filled-in corners. Pick the color that you want to outline, then start the outline on that color.

Anti-Aliasing

This tool helps prevent the optical illusion that often occurs when two highly contrasting colors are butted up against each other. This tool also helps eliminate the saw-tooth effect appearance of diagonal lines in low resolution graphics.

Merge/Blend

This tool allows you to superimpose another image over your current image. You can control the transparency of the image that you are importing. Be sure that the image you are importing is smaller than your current image. If you try to import an image that is too big, the operation will fail.

Brighten/Darken

You can use this tool to brighten or darken lines or rectangles in your image. Be sure to say NO to the prompt that asks you if you want the operation to affect the whole image and its palette. You never want to change the palette since this would make the image incompatible with all the other images in your game.

Smooth Image

This is one of the coolest special effects that GCSPAINT has to offer. You can smooth all the pixels in your image, or you can apply the tool to the edges of your image. You can control the intensity of the operation: light, medium, heavy, and extremely blurred. Other stunning effects include negative image, grainy glass, and double image. The pixel wrap around feature is especially useful when working on wall panels.

Contrast

There are times when you want to adjust the contrast of your image. This tool is just about as easy as fine tuning the contrast button on your TV!

Rectangle tools

These tools include copy, move, overlay, and erase. In addition you can flip, rotate, and resize your image.

Zoom

There are times when you need to move in close to work on the details of your image. Zoom makes this possible.

The best thing about this tool is that most GCSPAINTS's features can be applied while in zoom mode. It is also possible to save the zoomed in part of your image as a separate file. A prompt warns you not to save the zoomed section under the same file name as your original image, so you don't overwrite it

Oops

You can cancel just about any operation in GCSPAINT by selecting the oops icon on the main menu. Usually prompts warn you that certain operations can't be undone. Another way to cancel an operation is to click your right mouse button.

Summary

You'll be surprised by how quickly you become a proficient user of GCSPAINT. Its palette and sizing features make it ideally suited for preparing images to use with the GCS. Be bold. Experiment. Try out the super special effects. You'll be churning out great looking artwork for your game in no time!

LIGHTING EFFECTS

Lighting registers

Like universe registers, there is also a group of lighting registers that set parameters for the fading and object-lighting effects. Some registers affect every object that does not have the 'do not fade' attribute. Others affect only objects that have the 'flicker' attribute set.

Here are the names of the registers. Each lighting register is referenced by a number. For example, lighting register 0 is called

lgtfstart. These are NOT universe registers. Universe registers are different. The only way to set these lighting registers is with an animated object command. Thus, if you want to change the lighting effects, you must create an animated object that issues the 'set lighting register' command, and you must place one of these animated objects in your level.

Meanings of the lighting registers

#0 lgtfstart	Distance in units that fading begins: (0-8191) Default=400
#1 lgtfamount	Power of 2 that controls fade quickness (6-9) Default=8 6=quickfade(dark cave), 9= gentle fade (outside)
#2 lgtfmore	This distance is added to an object's distance before calculating the shading if fademore attribute is set. (0-8191) Default = 450
#3 lgtrate1	This number controls how fast the smoothly cycling panel goes from white to black etc. There are three cycles all of which can be independently set. Default= 4
#4 lgtrate2	This is for cycle #2 Default= 4
#5 lgtrate3	This is for cycle #3 Default= 4
#6 lgtflashrate	This number sets the rate at which the flashes occur, (0-255) Default=64
#7 lgtflashduty	This determines the fraction of time spent with light on. Default=64 0=always off, 255=always on.
#8 flrspeedx	This makes a textured-mapped floor move in one direction. Its speed has a sign. Default=0
#9 flrspeedy	This makes a textured-mapped floor move in another direction. Its speed has a sign. Default=0

How to do lighting effects

Ordinarily, objects will fade to black as they go into the distance. They don't have to fade to black, as you can set the fade color when you click on the palette icon in the GCS menu. This operation will generate fading tables for fading objects. You can have a different table for each level. This fading system will treat all wall sections the same. They will all fade as distance increases.

If you don't want a certain object to fade, set the attribute of the object to be no-fade with the 'EDIT ATTRIBUTES' command. This is useful for lamps, lights, fires, torches, glowing keys, and other objects that have their own light sources.

To accentuate the 3D effect of walled structures, north-south walls should be shaded differently than east-west walls. A quick look at DOOM or Wolfie will show you how this increases the 3D feel of a level. The GCS will do that automatically for you. First, select the walls in your level. Then, use the 'EDIT ATTRIBUTES' command from the Object pull down menu. Click on the button next to 'Set Wall Lighting By Angle'. You'll notice how great corners look now. The wonderful thing about this feature is that you get this great effect without having to create slightly darker wall panels by hand in GSCPAINT.

For special lighting effects, there are also ways to create panels that flicker in brightness or smoothly pulsate. There is one flicker system, plus three smooth cycles. All objects set to use the flicker system will flicker at precisely the same time together. There is no way to get an object to flicker differently than another object, unless you make it an animation file, and have multiple artwork panels. You have three separate smoothly cycling systems, which by default all run at the same rate, but out of phase by 1/3 full cycle. You can independently change the rates of cycling using the rate lighting registers described above.

Specify the lighting system for each panel by putting numbers in the red boxes that appear when you select one (and only one) object with the 'EDIT ATTRIBUTES' command. The third red box controls lighting. Please make sure the 'automatic lighting override' option is off, or else your lighting effect command number will be ignored.

The lighting commands are number values. The bytes are a combination of command value and a +-16 numerical value. To specify what system to use, you multiply the command number by 32 and add the numerical value. The commands are listed below.

```
0 * 32 = 0      ; don't modify shading with any system.
1 * 32 = 32     ; modify shading by n fade levels
2 * 32 = 64     ; modify shading by n*cycle 1
3 * 32 = 96     ; modify shading by n*cycle 2
4 * 32 = 128    ; modify shading by n*cycle 3
5 * 32 = 160    ; flash  to relative shading number n
```

The above numbers are lighting command numbers. These are set for each object that you want to have special lighting. Each time you use one of these lighting command numbers in an object, you must also use a numerical value. For example, I may want the brightness of an object to pulsate. I use command '2' which is 64 after I multiply by 32. If I just put a 64 in the third red attribute box for a wall panel, then I have forgotten to put in the numerical value. A pulsation with a depth of 0 fade levels is not very interesting. In fact, it is equivalent to setting the "DO NOT FADE ATTRIBUTE." To get the pulsating effect, I must add a numerical value to the 64 before I put it in the red box. If I want it to fade to half brightness, I add 8, which will take the brightness down 8 levels, and back up to full brightness as the wall pulsates. So I put a 72 in the third red box. Typically for maximum effect, you want to use a number n of 15. For the flickering, you usually want the lighting panels to have normal shading when off, and 0 shading when flickering on. So for those, you specify an n of 0. In the flickering on, fading will always be 0 (no fading), and off will be normal-fade + n. Never use an 'n' value greater than 15, or you may get unexpected results. Only use n=0 to 15. The larger the number, the darker the cycling will go.

Flickering can give a really nice effect. You can make very white lights, and put them all over. Set their second to last parameter equal to 191, which flickers from 0 to the last fade level. Then put the wall sections which face the lights to 160 in their second to last parameter columns. Then when the lights are off, the wall fading will be normal, and when lights flicker on, they go to full brightness.

We don't recommend trying to figure out exactly how the numbers work for these effects. Instead, just read these guidelines and start throwing numbers between 0 and 255 into the third red box, and experiment. This can be confusing, because we talk about both lighting registers, and these lighting commands in the same chapter. Do not get the two confused. Lighting registers control the lighting effects for the whole level, and don't apply to any specific object. The lighting commands are numbers that you set for walls or symmetrical objects to give those objects special lighting.

MAKING THE FINAL GAME

After making all your levels in your project, the final step is to hook the levels together with stairwells or other things. Then click on 'Make Final' to create the game. It is very important to read this chapter carefully. If you don't do these steps just right, your final game will not function properly.

Connecting the levels

To make your final game, each of your levels must have a unique 'level #'. In general it makes sense to set the level that the player starts in level 0, and the next level 1, and so on until the last level. In more complicated level connections, it doesn't have to be linear. This means you might have level 0 connected to levels 3, 5, and 10. Then there really isn't a sequential progression of levels, since the user is going all over the place. Each level needs to have a unique level number before you can make your final compiled game.

Warp-to Points

Traveling between game levels is done through 'Warp-To' points. A warp-to point is merely a place where the player will appear. These points are set using the Warp-To Point Manager from the Level pull down menu. You can put as many Warp-To points in a level as you like. You assign a name to each warp-to point after you set it with the mouse.

Once a warp-to point is defined, then you can make the player travel to that level and position anytime. You can cause this to happen by using platforms, or by having an animated object execute the 'warp-to point' command. Normally, the passageways between levels are done with a simple invisible platform placed around a stairwell wall piece. When the player walks within the invisible boundary of the platform object, he is whisked to a different level and position.

Please note that the warp-to point is only a destination, it is the platform that makes the level switch happen. To make a connection between two levels, you must set both a warp-to point in the destination level, AND a platform in the original level.

!!! NOTE!!! This does NOT work in test mode. All warp-to platforms or animated object command will be ignored in test mode. These will work only in your final game that you start with the 'GO.BAT' batch file.

It will be a good idea for you to keep track of all the connections between levels on pencil and paper. This is very important. Do not try to keep the level numbers and the warp-to point names in your head. List each of your level names, and write down the name and level number on a list. When you have assigned each level a number, and have written it down on paper, it is time to assign connections between the levels.

Internally, the GCS makes a table when you use the warp-to manager from the LEVEL menu. Each warp-to point has a name typed in by you. Warp-to points are a specification of a level number and a position. For example, in a game I want the user to start in level 0, walk up to a staircase, and then do a level switch to level 7. So assuming you have already used the set current level number to each of your levels, then you can go about setting warp-to points. So you open level 7 of your game. Then, you click on the 'Warp-to Point Manager' menu choice in the LEVEL menu. Then choose 'add', and then click on the spot you want the player to appear. Then a text box pops up which asks you to choose which direction the player should end up facing. North is up. Then it asks you to type in a reference name that you make up. Choose a descriptive name like 'Back into the lobby from stairs', so you will know which warp-to is what.

After you type in the name and hit return, you will see a red dot in your level. This is your warp-to point. It is not an object in your 3D world. This is only a marker on your screen to show you that there is a warp-to point. If you zoom in, you will see that the dot has a spike sticking out to show which direction you would be facing. Because it is not a really 3D object, the 'box and question mark' icon will not identify it. If you would like to see which warp-to point it is, then bring up the warp-to manager and click on 'identify warp-to'.

Thus, if a platform is set to warp the player to the lobby, he will be warped to level 7 in the final game. He will appear at the place you clicked upon.

So now you have set one warp-to point. To make it a functional passageway between levels, you must make a platform object in level 0, and place it right in front of the staircase wall. Make sure that the platform sticks out at least 150 cm from the wall. When you make the platform, click on the 'Warp-To' box. Then when you click on 'ok', you will be given a list of the warp-to points to choose from. Select the one you just made. You have formed the connection between level 0 and level 7 of your game.

Note: Platforms can only do one function at a time, so only press the 'Warp-To' button, and not any of the others. If you click on more than one function for your platform, the platform might not work.

If you want the connection between the level 0 and 7 to be a two-way connection, so the user can come back the same way, then you must make another level 0 warp-to point. You must make a platform which sends the user there in your level 7. So to set a warp-to point in level 0, you need to open level 0 and then click again on the warp-to manager from the LEVEL menu, and assign your new warp-to point a name. The trick is to make sure that you don't put the platform in level 7 where the user lands when he

is teleported in from level 0. This mistake causes an endless loop. The perplexed user bounces back and forth between the two levels indefinitely.

Important considerations

No Level switches in test mode.

The only way to test your inter-level connections is to click on 'make final'. Then, exit the GCS, and run the game from the target directory. Level switch platforms and animated objects that give the level switch command will be ignored when trying out your levels in test mode.

Make Final will NOT work unless you have specified a game start position.

Bring up the warp-to manager, and click on the command to set the final game start position. Don't get this confused with the 'set player position' command on the Level menu. That command is just for testing the 3D world.

After you specify and name the final game starting position, it will show up as a yellow dot with a spike. You can obviously only have one game start position. This initial starting point can be on any of your levels. The player does not have to start on level # 0.

Make Final will NOT work unless you have tested all your levels.

When you click on 'test level' to try out your level, several files are created in your project directory on your hard drive. The make final feature uses these files to compile all your .vgr files and other files into compressed binary files. The upshot is that unless you have clicked on 'test level' for each level, the make final command will fail.

When you make changes in any level in your project, you must test the level in the 3D world. Otherwise, when you click on 'make final', your changes will not be reflected in the final product.

If you change the level number of a level using the 'SET CURRENT LEVEL NUMBER' command, you must click on 'level test' and enter the 3D world and come back. Otherwise, that level will not be stored correctly. If you make a mistake and give two of your levels the same number, it will get confusing. Only one of the two levels will make it to the final game. The level that you tested last is included. Its twin is bypassed. Then in your final product, any warp-to points that were meant for the missing level would take the player to inappropriate spots, since the warp-to points weren't meant for that level.

When you click on 'Make Final', only levels that have warp-to points assigned to them will be included in the final product. This makes sense, because if there is no way for the player to get to that level, there is no reason to include that level in the final product.

Setting the Game Start position

The 'set player position' command in the LEVEL menu is only for using with the test command. This command does NOT set where the player starts in the final game. These positions are ignored in the final game. In order to set the starting position for the player in the final game, you MUST use the command called 'Set Final Game Start Point' from the WARP-TO MANAGER MENU. This will set the location of the player when he starts up the final game using the GO.BAT batch file.

The correct sequence of preparing for Make Final

First get you levels all the way you want them. You can set your level numbers as you go, or set the level numbers later when you are ready to do your warp-to points. However you may want different levels to fade to different colors, or maybe you want the game to use a different BACKG??.VGA 3D background for different levels, it may be more sensible to set your level numbers as you work. Of course, you will want to write down the level numbers on your precious sheet of paper!

Then you will want to go through all your levels to figure out where you will want players to enter the levels. Set the warp-to points, and write them down. Then go through all the levels and put in the level switching platforms. Use your list to get the correct warp-to point names. If you set a platform to a nonexistent warp-to point, you will get a critical error when you go over that platform in the final game. Don't forget to hit 'test level' after setting each warp-to point. If you don't test each level in the 3D world after changing it in any way, the changes don't get put into your final product!

Don't forget to set the final game starting point from the warp-to menu. If you forget, the GCS will refuse to make final.

After this, you are ready to 'make final'. The machine will be busy for a long time, especially if you have a lot of levels. When it is done, you will simply be back in the GCS, as if nothing happened. It is finally time to try your finished game.

Trying out your final game

To try out your final product, you exit the GCS, and then change directory to the target game directory. Then you run the game by typing ONE of the following command lines...

Go.bat command line options

go	(for no sound)
go s1	(for sound blaster sound; letter “s” and number one, ***NOT*** letter “s” and letter “l”)
go s1 t	(for sound blaster sound, and midi music)
go s1 t j	(for sound blaster sound, midi music, and joystick control)
go s1 t m	(for sound blaster sound, midi music, and mouse control)
go s1 t m g	(for sound blaster sound, midi music, mouse control, and god mode)

You will notice that the game loads much faster than 'test level' does.

What to do if there are problems

Type "s one" (s1), not "s" and the letter "L" (sl). DON'T put slashes or dashes in front of the command line options!

If your final game doesn't work right, don't panic. If you are getting a critical error message, look at the critical error section of the manual for some guidance. If the level to level connections aren't working, then you can always redo them. Use the 'box and question mark' icon to tell you what each platform is set to do, and examine your warp-to points. You can overwrite their current values by erasing and replacing platforms, and using the warp-to manager to reset your warp-to points.

If you feel that your warp-to points are messed up beyond repair, then exit the GCS and delete the file WARP_TO.DAT, and GAMESTRT.DAT, and then start putting your warp-to points in again from scratch.

!!! NOTE !!! if you remove a warp-to point, it is up to you to also remove any platforms that direct the player to the now-absent warp-to point. Warping to a deleted warp-to can cause unexpected results, or a critical error.

Common problems in the final game

If you have more than one weapons directory, make sure ALL your levels in your project use the same weapons directory. If not, the player's weapons and inventory items won't exist in some levels. The critical errors would be unpredictable.

If you have music in one of your levels, there must be music in all your levels.

If you use the lighting registers via animated objects, they carry over from level to level. If you change one level from the default values, you must put another animated object in the neighboring levels to restore the lighting when you come back from the lighting-modified level.

You might find it helpful to print the file theaters.txt from your target data\ subdirectory. The first column is the warp-to point number. The third column is the level number. The next three numbers are the x, y, and z coordinates in the 3D world where the player should go to.

If weird things are going on, like levels are missing in your final game, or you are appearing in totally unexpected places, a common mistake is that you have given two levels the same level number. The GCS will not complain about this, but the final game will never work correctly if you do this. This will not normally generate critical errors, but can make level switches to unpredictable places.

Another strange thing that can happen is that floors and ceilings work fine when you TEST LEVEL, but are missing or

incorrect in your final game. This is because the current floor status is saved in the warp-to points. This means that if you want to fix this problem, you must clear your warp-to points in the offending level and place them again.

Legal notice of what you may distribute

When you use the 'Make Final' command, the GCS makes a directory on your hard drive with the same name as your project. You may distribute all the files in this directory and the data\ subdirectory. These files comprise your game. The .EXE files in that directory are copyrighted by Pie in the Sky Software, but like shareware programs, you have implicit permission to distribute them in unmodified form. The same goes for all the other files that the unmodified GCS puts in that directory when you 'Make Final'.

When the user copies the game files from your floppies to his hard drive, make sure that he preserves the directory structure. This means that the DATA\ subdirectory and its contents must be contained within the main game directory when he types the go.bat command line to start the game.

PKZIP

Pkzip is the program everybody uses to package files for distribution on networks, or floppy disks. It is a shareware program, that you can download for free from almost any network or BBS system. For more information about PKZIP, you can call the PKZIP BBS at (414)354-8670 with your modem. It is available also on America Online, Compuserve, and other national networks too. If you actually use PKZIP to package your game files, you are obligated by the honor system to pay a small fee to PWARE, inc.

If you use the industry standard program PKZIP to put all your games in one big compressed file for upload/download, make sure you zip with the '-rP' option in order to preserve the DATA directory. If you forget this, then when the user unzips properly, all the files which are supposed to be in the DATA directory will just be thrown in the main directory with the .EXE files. For example, if you are zipping up the DEMO game after making final, do the following:

```
cd \demo
pkzip -ex -rP demo.zip  (the -ex just tells pkzip to use maximum compression.)
```

The bad news is that even when you properly use the '-rP' option, the user must use the '-d' option when using pkunzip to unpack your game onto the hard drive. So in addition to you using the '-rP' option when zipping up your game, you must ALSO make sure that your users use '-d' when unzipping your game. So in your README.TXT file make sure you make it clear to your customers that if the game gives them Critical Error #188, that they probably need to unzip the files again with the '-d' option.

So the users should do something like this for example:

```
cd \
mkdir demo
cd demo
pkunzip -d a:demo.zip
```

Putting your game on floppies

Your game will probably be too big to fit on one floppy disk, so you must put the files on multiple disks. You could do this by copying the files one by one with the DOS copy command, and then having your users make the data directory and copy the files into the proper directories, or you could use pkzip to have all the files and the data directory stored properly so the user doesn't have to deal with anything but remembering to type the '-d' option.

So to put your game on multiple floppies using pkzip:

```
pkzip -ex -rP -& a:\demo.zip
```

CUSTOMIZING OTHER PARTS OF YOUR GAME

Making shading effects: fog & dark

The most common color to fade to in the distance is black. However, you are not limited to fading to black. For a nice fog effect, you can also fade to grey. You can even fade to different colors on different levels of your game. Before you can run the 'test level' command in the game, you must click on the palette icon to make a palette fading table. This is just a big table of color values that the 3D game engine uses as your artwork fades into the distance.

When you click on the palette table icon, the GCS runs the program PALTABLE.EXE. However, before the GCS calls paltable.exe, it requests for you to choose a color from a palette box.

Although you can pick any color, you make smart choices for the fading to look good. When the player is very close to a wall or other object, the colors the player sees are the same colors you saw in your original artwork in the paint program. However, as the player moves away, each pixel in the panel must be replaced by a darker version of the original color of that pixel. The darker color must be taken from the same set of 256 colors that the project uses for everything. If there is not a good choice for a darker version, the 3D engine must choose something that isn't very close to the original color. It is up to you to make sure that you are attempting to fade to a color that has a lot of shades. For example, you wouldn't want to be fading to a bright purple if you are using the palette $rp9a for your project. This is because there are almost no purples in the palette. All the artwork would not fade gracefully to purple as the walls receded. There would be big color jumps.

When you run the paltable.exe program from the GCS, it creates a file called p??.tbl, and palt??.tbl in your project directory. The "??" indicates the level numbers. For example, a level set to level #5 would have two .tbl files, p5.tbl, and palt5.tbl.

These are the fading tables for your levels. The p5.tbl is the normal one, and the palt5.tbl is the table used for objects that have the 'fade more' bit set. It is possible to copy the .tbl files around using DOS commands. You may want all the walls that are set to 'fade more', to actually use a different final fade color. We can't think of any reason to do this, but you might want to keep it in mind for a special effect.

Using a custom palette

If you don't want to use the palettes that come with the GCS, you can create your own palette directory. First, you need to save the palette with GCSPAINT. Then make a directory with the same name as the palette, with no extension in the main p3d directory. If you call your palette mypal.pal, then you would create a directory called:

c:\p3dgcs\mypal\

In it you put your palette file, and a tiny description file with two lines of 11 characters on each. This file should give a very short description of your palette.

You will need an object library in your palette directory. So start a new project with the open project command from the FILE menu. Your new palette should show up on the list of choices now. Choose it for your new project. Once you are in the GCS, use the FILE command to open a new object library. Give it any name you like, and it will be created. Import some artwork into it with the library import/modify command from the "OBJECTS PULL DOWN MENU."

You will also want to make at least one guard set and one weapon set in your new palette directory. The GCS will copy the default set over, and match them to your new palette. If you keep getting critical error 184, it means that one of your windowed walls or symmetrical objects in the weapons or enemy directories got matched to all black (color 0). Having a symmetrical or windowed object that has no non-zero pixels is illegal. You will get errors if the palette matching sets all the pixels in a .VGR file to color number 0. The most common culprit for this is the .VGR file called invisble.vgr, which is in your weapons directory. Some of the other small dark pictures can do this also.

Using a Custom background textures

When you don't use a textured floor and ceiling, the default texture may be inappropriate for your purposes. When you don't specify a floor or a ceiling for a level, the floor is drawn with a 'jiggling' fixed picture. This picture is in the file backg??.vga. The ?? indicates the level number. This number is set with the 'set level number' command in the LEVEL pull down menu.

DO NOT run GCSPAINT from the paint icon in the GCS to do this. Instead, exit the GCS, and call it up from the DOS

command line by typing GCSPAINT. Once you edit one of these .vga files, you might have to load one of the palette files in your project directory if the colors are all messed up. This is normal. Just click on the palette icon in GCSPAINT, and chosen 'load palette.' You will also want to say NO to the match palettes question, or it will try to preserve your messed-up colors by matching to the new palette. If you run GCSPAINT from the paint icon, you can only save as .VGR files. You need to save these back??.vga files in the .VGA format, which is slightly different from the .VGR format. By the same token, if you want to edit your wall or character artwork, you must start up GCSPAINT from the paint icon, because you want to save your images in the .VGR format.

The background texture jiggles when the player moves for a good reason. When playing a 3D game, your eye will pick up the texture of the ground and use it as a constant reference point. If the ground is drawn as a nonmoving picture while the player moves, it creates the impression that the trees and walls and buildings are zooming around on top of the ground, while the player does not move. This is a very disturbing effect. If you do not like the jitter, you can always put solid horizontal bands of shading in your backg??.vga file. Then although the image will be jiggling, it will be totally undetectable, because there are no features moving back and forth. Note that the above discussion is moot if you have selected a texture-mapped floor for your level. We are only talking about the case where you do not want to put a repeating 3D textured surface on the floor. In this section, we are talking about the constant flat picture that is put on the screen in place of the floor when you do not specify a floor.

Use GCSPAINT from the DOS command line (do not try to edit .VGA files with GCSPAINT when you call it from the icon.) Instead, leave the GCS, and type GCSPAINT to start it up. Then switch directories to your project directory, which should be a subdirectory of the engine directory. Then edit backg40.vga, or whatever backg??.vga you want.

When editing the backg??.vga file, is very important not to change the size of the .vga file. Also, the top row of pixels in the backg??.vga file is repeated over and over again for the whole screen top. So, if you want a rough green floor and solid blue sky, then make the top row of pixels in backg40.vga solid blue, all the way across. The rest of the backg40.vga image should be the rough green, just the way you want it to look on the screen. When the game is run, the first line is repeated again and again all the way from the top of the screen to the horizon. This represents your sky. Then, the rest of the backg40.vga file is displayed. This portion of backg40.vga represents your grass.

Don't put a mixture of colors in the top row of pixels in the backg??.vga file. Anything but one solid line across the top of the backg??.vga file will show up as vertical lines and bands. Limit yourself to one solid color for the top line.

The next time you test your level, the new background should be there. Usually, make your top line black, and your texture go to black too. This is because if you have your fading set to fade to black, then you will want the horizon to be black too. If you specify a floor, but not a ceiling, this backg??.vga texture will be drawn upside down on the ceiling.

Customizing the hit point guy

Run GCSPAINT from the command line, and the load the .VGA file that has the name htpt.vga in your project directory. Do

NOT change the dimensions of the hitpoint guy, not even by one pixel. You will see 20 frames of damage increasing. These can be edited to any pictures you like, although you cannot change the dimensions of each frame. If you do so, then only parts of your hitpoint frames will show up in the damage window. Also, do NOT save these .VGA files with a palette, or the game will not function properly at all. If you save it with a palette, the game will be stuck in a slow process of fading in and out from black constantly.

Customizing the game play screen

This is just a .vga file called BACKDROP.VGA. Run GCSPAINT from the DOS command line so you will be saving files in the .VGA format. Edit the file backdrop.vga in your project directory. Leave it as 320x200, and don't save the palette. You can move the position of the little numbers that keep track of the score and so on, but the rest is not movable. You cannot move or resize the hitpoint guy, the radar, the weapon icon, the inventory icon, or the inventory boxes or the viewport.

You can change the compass, the LED numbers, and the GOGGLE by editing compass.vga, numlist.vga, and mask.vga images. Mask.vga is the nice border that has rounded corners that go around the viewport. You can make the mask any shape, as long as it doesn't change size. If you want a goggle or periscope effect, mask.vga in your project directory is what you want to edit. Please note that this MUST be SOLID color, not a picture. Even if you try to put a texture in, the mask will still be drawn in a single solid color.

Customizing the Help and Pause screens

Not surprisingly, editing help.vga, and pause.vga in the project directory will accomplish this. Don't forget that GCS has to run from the DOS command line before you can save in the .VGA format.

Customizing the number readouts

Edit the file READOUTS.TXT in the project directory. This allows you to put universe register values up anywhere on the screen. This can enables the player to keep a score during game play, for example. For a description of the few universe registers, which have dedicated purposes, see the chapter on universe registers. Print out or read the readouts.txt file itself for more information.

Menu for the final game and SAVE/LOAD

At present, you only have go.bat to run your final game. To answer the need for a proper menu with save/load and nice entrance and exit screens, we are developing an add-on product for the GCS users. It will feature .FLI animations in addition to nice still pictures. It will be possible to make various different endings for you game, determined by universe register 127. You can show a different outcomes too. There is even the possibility of halting the game play for a .FLI or still shot, and returning to the game play. This can all be controlled by an animated object. Call us for more information about this new add-on product.

NOT ENOUGH MEMORY?

You can get DOS to tell you how much available memory you have by typing MEM at the DOS prompt. Look at the line that says 'largest available program size.' That is the number that has to be 600000 or greater. The GCS needs 590k to 600k of available DOS ram, and about 2.5 megs of XMS or EMS ram.

The install program detects memory configurations that could cause problems for the GCS. If this is the case, you can either make a special boot disk for the GCS. You can try running the GCS anyway, bypassing the configuration warning, or you can read about configuration issues from helpful text screens.

If you have played DOS games that required you to make floppy boot disks, you might want to just try booting from one of those. Maybe one of those boot disks will start DOS with enough free RAM, and XMS or EMS ram so you won't have to go through the rest of this.

You can also purchase memory maximizing programs like QuarterDeck's QEMM or Qualitas's 386MAX. These memory management programs have lots of documentation and they automatically maximize the 640k area. If you buy one of these programs, you will discover why we can't tell you everything you need to know about DOS memory tricks. The manual for QEMM has a lot more pages that the entire GCS manual. While this may disgust you, it is not our fault. Buying one of these programs will cost you a few dollars, but it might be well worth it if you don't want to spend hours and hours tweaking your config.sys file.

Freeing Up Available Memory with CONFIG.SYS

Everybody has 640k DOS ram, but both DOS itself, and device drivers, take up space in that precious 640k area. In general the things that use up that space get loaded up in your config.sys. Config.sys is a text file in the root directory of your C: hard drive. It is a sensitive set of commands that can easily become messed up. If you start changing lines in your config.sys file, you may find yourself in a situation where your machine won't boot up at all. You may also lose the lines that load CDROM, scanner, sound card, and mouse drivers. We recommend that you NOT modify your config.sys unless you know what you are doing. Do not attempt to fiddle with your config.sys file unless you have made a back-up boot disk. Save your original config.sys on this boot disk. Make sure you have a fail-safe way of booting your computer from a floppy and restoring your original config.sys and autoexec.bat files.

We can't stress enough that you read your DOS manual about it before experimenting. This is a great way to learn about how your computer functions, but it can be very frustrating and time consuming. Since each computer is uniquely configured, we cannot tell you exactly how to increase your available memory, but we can point you toward more info.

Here are some topics to study in your DOS manual:

MemMaker
HIMEM
EMM386
Loadhigh (lh) command
High memory area (HMA)

Most of these swap stuff out of the 640k area into 'high' memory. If you see lines in your config.sys file like 'DOS=HIGH', or loadhi.sys, then you can see things are being put in high memory instead of taking up your precious 640k RAM. So if the GCS is complaining that you don't have enough DOS ram, then you must massage your config.sys file until enough stuff is loaded high, or not loaded at all. You need at least 600k or so free. If you feel nervous about getting into this, then find someone who knows how to modify config.sys files. Or, call a local computer business.

If you do decide to go ahead and tinker with your config.sys, we suggest that you make a back up of your original config.sys and autoexec.bat files. Put these on a bootable floppy disk and set the disk aside for safe keeping. If things go haywire, you'll be able to boot your computer, copy your untouched config.sys and autoexec.bat files to the root directory of your hard drive and restore your system to its original state.

If you opted not to make a boot disk when you installed the GCS, you can use the program BOOTDISK.EXE in the main P3DGCS directory. Or, you can make your own boot disk from scratch. Just use the format command with the option to make it a bootable floppy. Copy your config.sys and autoexec.bat files onto it. Go through the config.sys file on your experimental floppy line by line. Remark out or remove lines that load TSR or memory resident programs. Do the same with lines loading scanner and CDROM drivers. Use the proper DOS commands to load the remaining drivers into high memory. Use this floppy to boot your computer. Type MEM at the command line to see how much memory you have freed up. Keep repeating this process until the GCS can run without 'out of memory' errors.

Info about Available XMS or EMS ram.

The GCS can use either XMS or EMS memory. When you go into your 3D world, the game engine looks to see how much is there, and uses EMS if there is enough. If there isn't 2.5 megs of EMS, it will use XMS memory. IF there is no XMS, you will get a critical error.

If you have 4 megabytes of ram, you can configure your PC to run the GCS. If you have 4 megabytes, and the GCS is telling you that there is not enough available memory, there is probably something in your config.sys that can be fixed. There are two possible problems:

1. You don't have an XMS or an EMS driver loaded. -OR-
2. Some program like a disk cache program is sucking up your EMS or XMS ram.

Run the DOS program MEM. It will fill your screen with info about how much XMS and EMS ram you have. It will also tell you how much is used. If you don't have HIMEM.SYS loaded, or some other alternative, you will have 0 bytes of XMS and EMS. Look in your DOS manual for the sections on himem.sys and emm386.

If you have lots of EMS and XMS ram available, but all or most of it is used up, then a disk cache program like SMARTDRV.SYS is probably claiming all the RAM as its own. This would be on a line in your config.sys. A disk cache is a good thing to have with the GCS, so don't take it out, or many operations with the GCS will get slower. Tell the disk cache not to take up as much ram. For SMARTDRV, consult your DOS manual. For others, you must find documentation about how to change the line in config.sys to reduce how much memory used for the disk cache.

!!! NOTE !!! If all attempts to increase the amount of memory don't even come close, it may be because you have a 'Stacked Drive'. This means that all the data on your hard drive is stored in compressed form, and much of your precious DOS ram is being used up by the compression program to do this. Consult your stacker program manual about ways to possibly reduce the amount of conventional memory your stacker program is using up.

Why PC memory is so weird

To run the GCS, you need to have 590k to 600k of DOS memory available. When people get this message when trying to run the GCS, most are unbelieving. They say, "that cannot be, I have 16 megabytes of RAM in my computer!"

This is one of the less wonderful things about the DOS operating system. When you are running DOS, you are stuck with the limitations of the original IBM-PC computer which had a hardware limit of 640k of memory. The story that follows attempts to explain where all this complicated mumbo jumbo came from. It also proves to you that it isn't our fault!

Over the years, the computer hardware has improved incredibly. If cars had improved at the same rate as computers, the average car would have a maximum speed of 2000 mph, and go from coast to coast on a tank of gas! However, the DOS operating system did not keep pace with the hardware. When the 286 processors came out, Microsoft did not advance DOS to reflect the fact that you could now put more RAM in PC computers.

The price of RAM was falling dramatically, and the need for more RAM was great in spreadsheet and database programs, Lotus, Intel, and Microsoft came up with a kludge called EMS memory. EMS memory was a weird board that you could put an original IBM PC. It had memory on it, but it wasn't directly connected to the CPU like your regular memory. It worked by electronically swapping chunks of RAM in and out, similar to a record changer. Record changers were a solution to the problem of not being able to put more than 45 minutes of music on a vinyl record album. Thus spreadsheet and database programs could use more RAM than could theoretically be put in a PC.

So then the PC's had two types of memory. You had 640k of the regular 'conventional' memory, and then you had the kludge board with EMS RAM on it. With the 286, you could have megabytes of 'conventional' memory. So there was no longer any

need for EMS memory, right? Wrong. As we stated before, Microsoft decided not to update DOS to use any RAM above the 640k limit. Perhaps it was because they wanted the world to switch to Windows.

So many people went to the store and bought more memory for their 286's expecting to have more RAM available for their programs. Unfortunately, it didn't do any good. The only thing that memory could be used for was a program called 'VDISK'.

VDISK was a 'virtual disk' program. You could use your extra RAM above the 640k limit as a weird little hard drive that lost its contents every time you rebooted. You would do this because once you copied files onto that space, accessing them was very fast, especially when compared to the old 60ms hard drives of the day.

Since there was enormous pressure to find more RAM in software applications, some programs started to use RAM beyond 640K by accessing the hardware directly. This allowed the program to bypass the DOS operating system, which was pretending that RAM wasn't there.

This works fine, if you only have a single program that uses that RAM. It is usually the operating system's responsibility to keep different programs from using the same RAM. So without DOS watching over this 640k memory area, then software applications began to use the same memory for different purposes simultaneously. Like two trains on the same track, the results were disastrous.

However, there was a way out of this. Since the only IBM product that used this 'EXTENDED' RAM was this VDISK program, the DOS applications that used the extended ram started checking to see if VDISK was already using the extended RAM before grabbing it for themselves. So then programs also started writing stuff into the extended RAM, to fool other programs into thinking VDISK was installed. Thus the other programs would leave the extended RAM alone because the other programs would think that VDISK had already grabbed it. Thus, a virtual disk program had become a defacto standard for way for programs to use conventional RAM above 640k. This evolved into a standard called XMS.

If you remember EMS was a weird board that swapped RAM in and out like a record changer. However, those initial people were smart enough to control the hardware board through indirect software protocol. The application programs always just told another special program that came with the board what was desired.

This was a very good thing, because 386 computers could use your extended ram to emulate EMS memory. So when you load a program that uses the conventional RAM above 640k. It loads a program that makes it look like one of those EMS boards is in the computer. But, there is no such card. Instead, the extended ram is swapped in and out using special 386 protected mode software. To the application, it appears that RAM is still being swapped in and out from this installed electronic card. But in reality, extended RAM is just being 'remapped' using fancy tricks you can do with 386 instructions.

To use the conventional RAM above 640k, this EMS emulator had to pretend it was VDISK. So then you have a program pretending it is a hardware card, pretending it is VDISK, and remapping conventional extended memory by swapping in and out.

Nowadays, all EMS memory is done this way. When you add memory to your computer, it is all conventional memory to start with. But DOS ignores all the RAM but the first 640k. So programs that need more than 640k must look for XMS memory or EMS memory. XMS memory is the fake VDISK type of direct usage of the extra memory. EMS memory is a fake hardware board method of swapping blocks of RAM in and out.

This explains the 640k limit, as well as the origins of XMS and EMS. There is a lot more to the saga, but this is approximately what happened. So, if you are frustrated with the GCS because of memory limitations, please understand that we are just as frustrated and we sympathize with you 100%!

TOP TEN TROUBLE SHOOTING SOLUTIONS

1. The program complains that I don't have enough memory, although I know I have lots of it.

This is due to the fact that DOS doesn't handle memoryabove 640k consistently. Because of the complicated nature of RAM in DOS, we have devoted a whole chapter to it, cleverly named 'Not Enough Memory?'.

2. How do I add save and load my games?

The best solution is to buy our companion product, GCSMENU. The price is only $34.95, and you get a full customizable menu system. The menu system integrates VGA screen images and .fli animations, as well.

3. I put platforms down in my level, and warp to' points, but it never works. I go there and stand right in the middle of the platform, but nothing happens.

Level switching cannot occur when testing an individual level. You need to 'make final', and then run the final game as explained in the chapter called 'Making the final game'.

Also, make sure that you 'TEST LEVEL' after setting any platform. If you forget to test your level after making changes, then the platform will not actually be in your final game.

Use the '?' icon to check that your platforms are in the right place and are set correctly. If in doubt, delete your platform and replace it with a new one, but don't forget to 'test level' before making final again!

4. How I put more than one kind of enemy on a level?

Because all the different views required to represent the enemies, you would run out of memory space for artwork if more than one enemy type were allowed on a level. But don't despair, there are ways around this.

One solution is to make a stationary enemy as an animated object. Enemies created in this fashion can attack the player, and they can be vulnerable to the player's weapon also. This type of enemy would be made out of symmetrical bitmaps that always face the player. Thus, fewer artwork frames are necessary.

Also, you can split a 'floor' of a maze or building into several 'levels'. This would allow several different types of enemies in the same general area perceived by the player. In fact, you can have one room as its own game level if you which.

Call us about the upcoming GCS Professional Version.. This version of the GCS will handle the memory so all 4 or 8 megabytes of RAM will be available for your artwork, thus alleviating this restriction on the enemies.

5. When I save my files in GCSPAINT, they don't get saved, or they are saved as the wrong file type.

There are a few different things that could be happening. First, if you want to save your images a .VGR file, then you must have started GCSPAINT from the icon that is on the GCS layout editor screen. If started the paint program by typing 'GCSPAINT', then your files will be saved as .VGA files. If you save your .VGR file in the proper directory, it may not show up in your 'Import' list in the library manager dialog box. Be advised that although things you add to the list usually appear at the end, this is not always so. This is dependent on DOS file order on the hard drives, which is unpredictable. Look through the list carefully, and you can probably find it.

6. How do I change the damage indicator guy for my game?

Exit the GCS, and bring up GCSPAINT from the command line. You need to load the .VGA file called htpt.vga from your project directory. You will see that this image file is a tiled repeat of the damage indicator with varying degrees of damage in each picture. Feel free to change your damage images. However, do not try to change the dimensions of the little damage boxes, or the size of the whole htpt.vga image. If you do, the engine may crash.

7. How can I change what the floor/ceiling looks like when there are no textures specified for the floor or the ceiling?

You need to get out of the GCS and bring up GCSPAINT from the DOS command line by typing GCSPAINT. Click on LOAD VGA, and choose the ENGINE directory (in red on the screen). Then choose your project directory. Finally click on BACKG?.VGA. Replace the ? with your level number. If you want to change the non-textured floor/ceiling for level number 5, you must bring up BACKG5.VGA. If your level does not use a textured ceiling or floor, then the BACKG5.VGA image you see will be drawn as the floor of your game. If you tell the GCS to use a texture for the floor but not the ceiling, then the BACK5.VGA image will be flipped upside down and drawn as the ceiling.

See the chapter on Customizing for more information about this.

8. Why does my sound not work in my final game? Why doesn't my joystick work?

The joystick cannot be used when testing levels. As covered in the chapter entitled 'Making your Final Game', you need to add a 's1' to the go command line. go s1

You must also add a 'j' if you want joystick support. Be careful with this, because if the user does not have a joystick and you put the 'j' on the command line, a critical error will ensue. When you actually finish your game, you will probably want to use GCSMENU to create a way for your customers to choose sound on/off, joystick on/off, etc.

9. When I try to define written text for memos and animated objects to use, it never works.

You need to edit the file pstr.txt in your project directory, and then you must run the program pstr.exe on it to actually make your new text messages available for your game. The chapter on 'Text Messages' explains how to do this. Be very careful not to ignore the two periods and the backslash when you type PSTR! Also, make sure you understand how give each message in your pstr.txt file a different number when you are editing the file.

10. When I try to load my .GIF, .PCX, or .BMP file into GCSPAINT, the screen flashes, and it refuses to load my picture.

The most common cause of this is that you are trying to import a picture that has too many pixels in it. The largest image that GCSPAINT can handle is 200x200 pixels in size. If an image is more than 200 pixels high OR 200 pixels wide, the image will refuse to load. 200x200 is too big for a practical object in the GCS game. See the chapter in the manual on images for more information about these limits.

If you are absolutely sure that your image is small enough in number of pixels, but the image is still being rejected, then perhaps your .PCX or .BMP file is not the right type. Since the GCS is a 256 color game, all .PCX or .BMP files that you import must be 256 color also. 16 color files are not accepted, and neither are 16 million color versions of these files.

11.) Why is the computer beeping at me whenever I do anything with my level?

You are close to, or have already added to much artwork to your level, and memory is full. See the explanation of the the 'M' and 'W' meters in the chapter called 'Laying out a good 3D level.'

12.) Why do I need to type a 'b' to get my GCS to run? I have lots of memory! Please see the chapter on Memory.

13.) I can't even start up the GCS, because the first 'OK' button doesn't work when I click on it.

A very few mouse drivers have errors in them that prevent them from working with the GCS. If you suspect that this applies to you, then reboot your computer, and CD to the main GCS directory. Type the following command which will create a file which leaves a message for the GCS that tells it to try and get around the problem mouse driver.

COPY VV.PAL MOUSEFIX.DAT (If this doesn't work, give us a call.)

14.) The demo levels and Industrial Killers crash or give me critical errors. What's wrong?

Use a text editor to look at your config.sys file. You may be having troubles if you have FILES=8. Consult your DOS manual about this line and make sure you have a back up boot disk containing your original config.sys and autoexec.bat files. Try increasing the line to read FILES=30. Reboot your computer after making the change.

CRITICAL ERRORS

When you run the game engine, there is a lot of self-checking going on. When the game engine finds a situation in which it cannot continue, it stops and returns to DOS with a 'CRITICAL ERROR'. There is no need to panic when you get one of these. By far the most common reason for a critical error is that you have loaded too much artwork in your level, and you are running out of available DOS ram. There are various other mistakes that can lead to critical errors.

Occasionally, you might get critical error messages when running 'test level' from the GCS. When this happens, you will see a blue screen with the critical error number flashing on the screen. Note that number, and two other numbers. Then look for your error number in the section in this chapter called 'MOST COMMON ERRORS'. Often you can solve the problem by removing the offending object that you have just added.

Test your level frequently so you can easily deduce the root cause of your troubles! If you get the critical error when running your final game, then it will be left on the screen, and possibly left in the text file called p.out.

What to do if you are getting a memory alloc fail

There are two major errors that occur when there isn't enough RAM. One is a malloc fail, and the other is CE 199.

CE code 199 means that p.exe is refusing to run because it thinks there isn't enough DOS RAM. This check occurs despite what is in the universe. Right now, you need about 590k to avoid this error. When you get this error, a small text file called memfail is created that includes how much additional RAM is needed to get past the check.

If you are getting a malloc fail error, or perhaps a file open error, you may be running out of RAM. Look at the 'M' meter on the bottom of the GCS screen. If that meter is maxing out, then you have loaded too much artwork. Read the chapter on laying out your 3D game for more info about the 'M' meter.

If that isn't the problem, you are probably trying to load artwork that has too high a resolution.

Putting more objects into your universe does NOT effect RAM usage. The memory area for placed objects is fixed in size at 768 objects.

If you think you are trying to load too much artwork, remove all instances of a particular object. Preferably, make it an object that is high-resolution. The RAM that an object takes up is the width in pixels times the height in pixels. We are NOT talking about the 3D world size in centimeters that you set with the object library. We are talking about the pixel size of the image as seen in GCSPAINT. Each time you place a new object in your level, (and you have not placed any of those pieces earlier), you use a bit more memory. If you place a wall that you have already placed in your level, then no additional ram is used.

By the same token, say you had three telephone booth objects in your level. You attempted to reduce the ram requirements of the level by removing one booth. Wrong, this will not change the ram required. The game engine STILL has to load the picture of the phone booth to draw the other two phone booths that are in the game. Remember that it is the ARTWORK that takes up the space, not the objects themselves. In the above case, removing ALL the phone books from the universe will reduce the ram required.

If you suspect that you might not have enough available DOS ram at boot time, run the DOS command 'MEM' from the DOS command line. It will tell you how much available DOS memory is available for programs. And, it will tell you how much EMS and XMS you have. You need about 600K to run the GCS. If you don't know how to go about freeing up more DOS ram, consult one of your games that includes lots of info about a making boot disk and removing TSR's.

Most common errors:

Critical Error 13 — Back up your .wld file, and try removing the last few objects that you placed in this level.

Critical Errors 19 & 20 — You are probably trying to put a .vgr file in your level that has too many pixels in it. Resize the size of the artwork image using GCSPAINT. Also, it could be that there are too many black spaces. Read the chapter about artwork issues.

Critical Error 50 — See instructions for CE #13 above.

Critical Errors 53 - 66 — An animated object isn't working properly. Try removing every instance of the last animated object that you put in your level. Don't forget to back up your .wld file on a floppy first.

Critical Error 67 - 77 — A guard is not working properly. Back up your .wld file. Try selecting and deleting all the guards you have placed in your level, then try again with no guards.

Critical Error 79 — Here are the possibilities for a cause of CE code 79, nobdrop. Look at the value, and find the cause below: Value: Missing file:

9
31 HTPT.vga file. (subdir)
32 backg??.vga (subdir)
numlist.vga (subdir)

Critical Error 83 A problem is occurring with guards. See CE #67 above. It could also be corrupted .4th files in engine or project directory.

Critical Error 87 & 88 These should never occur. If they do, back up your .wld file, and start removing the last items you added to your level.

Critical Error 92 Trying to find a image file that isn't there. Maybe you deleted a file from your library directory?

Critical Error 95 Missing Midi file. Try setting the music again from the GCS musical note icon. ALSO, make sure your midi file is less than 32000 bytes.

Critical Error 116 See CE #19 above.

Critical Error 146 An object has been placed out of legal bounds. It is violating the white boundary rectangle that you see on the GCS screen.

Critical Error 148 - 158 Something is wrong with an inventory item. Back up your .wld file, and remove the last inventory items that you placed. Could also be a symmetrical object that accidentally has 'don't draw back' attribute set.

Critical Error 163 You forgot to run the ../pstr command from the DOS command line as described in the chapter on text. Or perhaps a string number is set incorrectly either in your level or in your pstr.txt file.

Critical Error 164 An icon is missing. It could be caused by missing weapon *fat.vga file, or a bad inventory object, or a symmetrical object with the 'don't draw backs' bit set accidentally. It could also be a picture memo type inventory item that is trying to load a .vga file that doesn't exist.

Critical Error 165 Problem with the Sound Blaster sound system. Could be a missing .wav file, or a problem with XMS or EMS memory. Try game without sound. Perhaps you have sound card that needs a driver in your config.sys before it is 100% sound blaster compatible.

Critical Error 167, 168 You have a symmetrical object with the object attribute 'don't draw backs' on by mistake. Is being taken as a pickupable object, but it has bad value for the icon handle. It fails with this error when drawing junk in the icon box for pickup. Check all your symmetrical objects with the 'edit

attributes' command, and make sure none of them have the 'don't draw backs' button on. Also see CE 164 above.

Critical Error 174 This means that a directory is missing. If you get this error when attempting to run your final game, then your 'make final' command has failed. There should be a directory called 'data' in your main target directory.

Critical Error 184 You have a windowed wall or symmetrical object that is all black pixels. Check all the symmetrical objects and windowed objects in your level to make sure they all have non-black pixels. Also check the file called invisble.vgr in your weapons subdirectory in the current main palette directory. Sometimes that .vgr image can go to all black when palette matched. If you keep getting it, check all the images in your weapons directory to make sure non of them are all black (color 0).

Critical Error 188 Missing objdef ??.txt file. Did you test your levels before making final?

Critical Error 199 Not enough DOS ram to run p.exe. Free up more DOS ram.

Critical Error 211 No EMS or XMS available, or not enough

Critical Error 247 Didn't test a level that has a warp-to point in your project before making final.

Here is a raw list of possible critical errors. In most cases, these will not be helpful, but it is included for those who would like to get maybe any hint at all as to what the problem is.

Raw Critical Error list

unerr1	1	unknown error	sptunk	17	weird type for drawkind
div21serr	2	unwanted sign in div21	nobkgrnderr	18	no background file found
div21oerr	3	overflow in div21	toomanyholes	19	too many holes in a bmplane pic
bmrrngerr	4	dynamic range error in bmrender	bmtoobig	20	bitmap is too big for hole anal
zbmlcfl	5	malloc fail for zbuf init	merrhole	21	mem alloc fail during hole alloc
zbnormerr	6	zbuf pointer unnormalizable	merrhptr	21	mem alloc fail during hole ptr alloc
objmemmalerr	7	objmem malloc fail	ssunx	22	expected species type 3 for symsegread
clmemmalerr	8	class list malloc fail	errnseq	23	too many frames in symsequence
objdferr	9	error in objdefs.txt file	objdfnf	24	objdefs.txt file not found
illspcnum	10	illegal species number	univfnf	25	val=0 univ.txt, val=3 pstr.str missing
typeallocerr	11	malloc failed for type allocation	sseqfnf	26	<MS>ymseq.txt file not found
badspc	12	bad species number in univ.txt file	ssbadspc	27	bad species # in <MS>ymseq.txt file
badtyp	13	bad type number in univ.txt file	ssbadfrmn	28	bad framecode number in <MS>ysseq.txt
objnfreer	14	attempt to create over non-free obj	ssbadtyp	29	bad type number in <MS>ysseq.txt file
badcorder	15	bad coord number in univ.txt file	ssmem	30	mem alloc fail in seqread
unexpspn	16	unexpected species in drawdeadwall	unxdss	31	unexpeted species type in drawsymseg

dssfrm	32	unknown frame code found in ssdraw
ssdbadtyp	33	bad sym type number in ssdraw routine
dssnbad	34	calling for illegal symseqdata element
ernofrm	35	symseq creation with 0 frames
ernoterm	36	last frame code must be 2
rssfrm	37	unknown frame code found in ssrun data
rssdbadtyp	38	bad sym type number in ssdraw routine
objmemexh	39	object memory exhausted
plyunx	40	expected species type 4 for poly in read
msngplyparm	41	a param in objdefs.txt file bad or missing
plynsides	42	too many/too few sides specified in ply
plymem	43	mem alloc fail in plyread
unxdply	44	unexpected species type in drawpoly
badwcorder	45	bad start coord number in univ.txt file
badwscorder	46	bad window size value in univ.txt file
pwnum	47	bad species number in readpolywall
rpwmem	48	alloc fail during polywall read
palramerr	49	alloc fail during palette malloc
copytbadtype	50	copycat's victim type no out of range
bmrmfail	51	malloc fail in bmreadfile
cbmrmfail	52	copycat malloc fail in bmreadfile
flunx	53	expected species type 3 for fluidread
ernfluid	54	too many frames in fluidsequence
fluidfnf	55	q.txt file not found
flbadcoord	56	bad coord number in .txt file
flbadspc	57	bad species number in .txt file
flbadtyp	58	bad type number in .txt file
flmem	59	mem alloc fail in fluidread
unxdfl	60	unexpected species type in drawfluidseg
dflfrm	61	unknown frame code found in fluiddraw
fldbadtyp	62	bad fluid type # in fluiddraw routine
dflnbad	63	calling for illegal fluidseqdata element
dflspcbad	64	species number bad in draw fluid
dflstybad	65	bad type number to draw fluid
flunexpspn	66	unexpected species at fluid draw
slunx	67	expected species type 3 for solidsegread
ernsolid	68	too many frames in solidsequence
solidfnf	69	.txt file not found
slbadtyp	70	bad type number in .txt file
slmem	71	mem alloc fail in solidread
unxdsl	72	unexpected species type in drawsolidseg
sldbadtyp	73	bad solid type number in soliddraw
dslnbad	74	calling for illegal solidseqdata element
dslspcbad	75	species number bad in draw solid
dslstybad	76	bad type number to draw solid
slunexpspn	77	unexpected species at solid draw
bfmemfail	78	stack alloca fail for mask bitmap
nobdrop	79	missing file, see * below for list
mdatamf	80	malloc fail during mdata init

bjtomdiv	81	divide by zero in objtomdata
pygerrmem	82	malloc failure in pygmyinit
pygerrinit	83	pygmy failed init
pfontmf	84	malloc fail during pfont init
pfontfnf	85	pfont.dat file not found
memhanderr	86	alloc fail in handle array alloc
handledup	87	duplicate handle name defined
handlechk	88	handle name differnt on pass 2
ernopchkf	89	No p.chk file found
ppalnf	90	No p.pal file found
horange	91	handle out of range
dwbmnf	92	bitmap file not found
erdapl	93	direct drawing of applique attempted
saplunx	94	wrong spnum in readapl
aplfnf	95	apl file not found
aplbadf	96	bad stuff in apl file
aplbadstf	97	bad 4 numbers in apl file
aplmfail	98	malloc fail in alpread
apldbad	99	bad number in apl file data (strt&len)
aplddbad	100	bad number in apl file data (data)
apsetfnf	101	apset file not found
badapnum	102	bad applique # in apset file
ernapset	103	illegal # of appliques in apset
apsterm	104	no terminating -1 in apset file
apsmem	105	malloc fail in apset
sapsetunx	106	bad species in readapset
badaphand	107	bad appl. handle in univ.txt
badapshand	108	bad apset handle in univ.txt
ernsolidf	109	too many frames defined in solid
slapunx	110	bad spnum in readsolidap
solidapfnf	111	solidap text file not found
slapbadtyp	112	solidap bad handle in text file
vertfail	113	no vertical retrace found
pygloadfail	114	pygmy load file not found
nopyg	115	No fload file name
htbufbig	116	too many holes in shaped bitmap
ptraor	117	pointer assignment out of range
ptrblknf	118	allocated block not found
ptrasser	119	too many ptr assigns to follow
optrseg	120	species and optrblk don't match
douboptrm	121	attempt to malloc optr twice
cncofile	122	could not create output blk file
badccode	123	fileblk has bad code
mxhandwr	124	handles not right size in file
nmblksbad	125	nmemblks not valid from blk file
badptrass	126	bad block number for ptr assign
rhandre	127	read handles has failed
handme	128	malloc fail during handles malloc
frnofus	129	nofus for object is bad

ubopen	130	binary universe file open error
ubread	131	binary universe file read error
badzcorder	132	bad z coord value in univ Z param
thtfnf	133	theaters.txt file not found
badent	134	bad entry number in theaters.txt file
dupentnum	135	duplicate entry found in theaters.txt
badobjn	136	bad object number in tht file
badunivn	137	bad univ file number in the file
badthtcrd	138	bad or missing xy or z coord in tht file
tnumnf	139	cant find entry in theaters
joycerr	140	joystick centering errror
nojoy	141	no joystick, where one was specified
erentnf	142	starting entry # not found in theater
flxfrmer	143	fluid attempt to xfrm to nonexit fluid
mfluidhan	144	error in reading fluid file, handle msng
gtn2	145	grenade bit set in object of species!=2
ctmcbound	146	ctmc caught value out of bounds
badflhand	147	bad handle in explosion # specification
badpnum	148	bad pnum in pickup type hand spec
badphandl	149	bad ptype handle numbher
mxpickwr	150	bad number of pickups in .bin file
inumobnd	151	bad arg to newinv
pickpupnf	152	no entry in pickuptypes for inum
invpnull	153	null invnumptr in newinv function
iblknfree	154	inventory block cannot free error
doorkbnd	155	door key value of out bounds
runcoob	156	bad runcode in basefunct
illurcode	157	bad usecode in pickup/drop object
rllurcode	158	bad usecode in run object
explerr	159	handle number 3610 must be explosion
vszmsng	160	vertical size number missing in objdef
hszmsng	161	horizontal size number missing in objdef
unexpspndr	162	dodoor found unexpected species type
unexistr	163	tried to print non-existant string
fdbmpfnf	164	file not found or read file in filedrwbmp
errpter	165	error in sound or xtended memory
req386	166	386 cpu required
bmaphoob	167	bmap drawfunction #hpts out of bounds
bmapvoob	168	bmap drawfunction #vpts out of bounds
errfats	169	vgafiles.toc or vgafiles.fat not found
fatserr	170	read error or loused up fats file
nopygmem	171	no pygmy memory available at present
extmalfail	172	malloc failed for compression memory
nocwd	173	no current path found
cncdir	174	subdirectory not found
erdiskspc	175	not enough free disk space to run game
erfrmv	176	univ?.bin file exists but can't be removed
statefbad	177	bad or messed up state.bin file
stateopen	178	can't open state file
diagferr	179	diag file error
erpcmd	180	can't remove p.cmd file
erstdout	181	cant redirect text output to p.out
erstrip	182	strip size not 1280 wide
dsbmer	183	bitmap too big for inventory drawing
fhshrink	184	shrunken block changed pointers on us!
ermouse	185	mouse requested but none found
bdamsz	186	bad size in damage artwork
idambe	187	idam bad value
objmfnf	188	memblkfile not found (objdef?.dat)
pygbiner	189	error in pygmy part of univ?.bin file
badheap	190	heap corrupted
nopygmyp	191	pygmyptr is null in bfindholes
ertime	192	program is past date of expiration date
gflspcbad	193	species number bad in draw gun
gflstybad	194	bad type number to draw gun
baddmachn	195	bad dma channel specifed in command
ersndnf	196	digitized sound never finished playing
pnocomp	197	no compile with with stripped p.exe
pdimsng	198	pygmy.dat file is missing from subdir
memsize	199	not enough ram at run time
errnscrn	200	no scrn.dat file found
emszbuf	201	ems zbuf error
pygchkerr	202	pygmy checksum error
unexpspp	203	unexpected species in prepdeadwall
unexpspf	204	unexpected species in prepfluid val=sp
badfltype	205	bad fluid type in fluidbump code
weirdptr	206	pointer doesn't have an offset of 4
memvfail	207	malloc fail in initvert()
badvert	208	bad vertex number in object data
vertbig	209	bad vertex number in object data
mctoobig	210	memchnk alloc size too big. (size)
xmemerr	211	no ems or xms available
nmemcerr	212	too many memchnks (nmemchnks)
emspgerr	213	ems page mapping error (hand,eerr)
xmspgerr	214	xms page map error (hand,xerr)
emsferr	215	ems free error (hand,eerr)
xmsferr	216	xms free error (hand,xerr)
mcmper	217	error in mapping of weapon ram
mcwper	218	error in writing of weapon ram
imemcerr	219	initialization err of memchnks
memcbeff	220	malloc file in initmemchnk
ddmcrm	221	memchnk remap after mdata failed
mcwrdf	222	mdata write fail in dodoor()
robodmf	223	malloc fail for robodataptr
pygicon	224	pygmy inconsistent block type
pygmapfl	225	mcmap failure in pygmywrite/read
pygmcwrt	226	mcwrite fail in pygmywrite
pygmchi	227	unexpected emtype in pygmy memchnk

pygemsfail	228	ems pygmy alloc failing with z option
pgyck	229	pygmy fails checksum
prmwc	230	temp mcwrite failed in readpygmy
prmrc	231	reading or freeing failed in readpygmy
deamcf	232	mcfree error in deallocation of memblks
pygnomtch	233	universe loaded pygmy fails chksum
transtmf	234	malloc fail in transtable malloc
sldpbadtyp	235	bad solid type # in prepsolid routine
sldbadtyp	236	solid bad type, but should be nice
rmemblkmf	237	malloc fail during readmemblk
mylbadspc	238	bad species in mylineofsight, val=species
qxybadspc	239	bad species in qxywall, val=species

INDEX

.4th, 92
.bmp, 29, 59, 62, 90
.fli, 83, 83, 88
.gif, 29, 59, 62, 90
.mid, 50, 50
.pal, 12, 81, 90, 95
.pcx, 29, 59, 62, 90
.tbl, 80-80
.txt, 30, 32, 33, 44, 53, 56, 58, 59, 78, 79, 89, 90, 93, 95
.vga, 10, 11, 17-18, 35, 57-59, 62, 66, 77, 82-83, 92-93
.vgr, 12, 30-31, 38-39, 49, 55-56, 66, 76, 92-93
.wav, 48-49, 61, 93
.wld, 92-93
$rp9a, 12, 12, 49, 52, 58, 80
320x200, 59, 59-60, 66, 83
386MAX, 84
400x400, 16, 62, 63
590k, 83, 86, 91
640k, 36, 61, 84, 88
64X64, 11, 61, 67
768 objects, 91
Adding animated objects, 30
Altitude adjustment, 21
Animated objects, 10, 15, 17, 26-27, 30, 38, 40, 44, 46, 54, 55, 71, 76, 78, 89
Animation commands, 40, 42
Armor, 51, 58
Artwork, 5, 8, 11-12, 25, 28-29, 31, 35, 38-39, 42, 49, 59, 61, 62, 65, 68, 69, 71, 73, 80-81, 88, 90, 92, 96
Backdrop.vga, 65, 66
Backg??.vga, 11, 77, 81-82, 92
Background textures, 81
BASICLIB, 49, 49
Bomb, 51-51, 54
Bootable floppy disk, 85
Branch if no guards, 44
Branch if shot, 42, 45
Buildings, 18-18, 23, 82
Bypass, 8, 87
CANCEL, 20
Can't bump into object, 23
CDROM, 84, 85

Ceilings, 11, 14, 17, 25, 61, 78
Choose an object, 12, 30
Cm, 11, 11, 16, 17, 19, 21, 25-26, 29, 31, 43, 44, 62, 63, 75
Color #0, 10
Common problems, 78
Compass.vga, 83
Config.sys, 6, 8, 93
Critical errors, 8, 23, 37, 40, 47, 59, 63, 67, 91-92, 94
Custom palette, 81
Damage, 8, 9, 35, 40, 89, 96
Damage player, 43, 43
Def_enem, 36-37
Demo, 7-7, 49, 61, 80, 90
Design new animation, 41
Display a text message, 44, 47
Document, 7, 51, 53
Don't draw backsides, 22-22
Don't draw if far away, 23
Don't fade with distance, 23
Doors, 22, 22, 32, 41, 60, 63
DOS, 6-6, 11-12, 16-18, 30, 32, 33, 36, 38-39, 61, 65-66, 81, 83, 85, 90, 92
Edit attributes, 51, 51, 54, 57
Editing, 9, 13, 16, 22, 30, 35, 38, 49, 54, 82-83, 90
Editing icons, 10, 10, 11
Einstein with an attitude, 34
Elevation, 10, 21, 41, 47
EMS, 39, 48, 83, 85, 92-94, 96
Enemy Characters, 15, 15, 16, 23, 33, 43, 44, 61
Entry points, 43
Erase, 21, 21, 48, 70
Exit screens, 83
Fading table icon, 10
Fat, 93, 96
File open error, 91
Find all, 22-22
First aid, 51
Fld????.txt, 39
Floor/ceiling, 10, 11
Floors, 14, 69, 78
Frame #, 31-31, 41-42, 45-46, 48
Game play screen, 25, 50, 54, 65, 82
Game start position, 76-76
Gas grenades, 51
Gas mask, 51
GCSPAINT, 5, 11, 12, 17, 18, 28-30, 39, 42, 56, 62-63, 65-66, 70-71, 83, 89-92
Generic ammo, 51
Go s1, 77-77, 89
Go s1 t, 77-77
Go s1 t m, 77, 77
Go.bat, 76, 77, 79, 83
God mode, 8, 8, 77
Googun, 54-56
Graphical problems with doors, 26
Grid, 11-11, 16, 19, 21, 26, 41, 60, 63
Grid icons, 11
Grid snap off, 21
Ground, 17, 17-18, 41, 45, 47, 56, 62, 82
Groups of objects, 10, 10, 21
Gunnone.vga, 65
Hand grenade, 25, 51
Handle, 48, 65, 90, 93, 96
Health, 9-9, 27, 50, 62
Health pack, 53, 53, 57
Help, 6, 6, 13, 14, 39
Help.vga, 83
Himem.sys, 85, 85
Hit point, 82
Holes, 14, 63-64, 94, 95
Horizontal size, 96
Htpt0.vga, 65
Icons, 9, 11, 16, 20, 28, 29, 41, 53-54, 56
Image file, 29, 29, 55, 57, 59, 60, 62, 89, 92
Image resolution, 60-62
Importing or creating new artwork, 28
Installation, 6, 6
Inventory, 8-11, 25-27, 37, 38, 52, 54, 56, 78
Inventory objects, 30, 54-55
Invisible blast, 43-43
Joystick, 77, 89
Key value, 25, 25, 96
Keys, 8, 8-9, 22, 25, 41, 57, 72
Layout, 10, 13, 16, 18, 23, 24, 47, 53, 57, 89
Left click, 13, 24
Legal notice, 78

Level Menu, 75-76
Level switches, 43, 75, 78
Library import/modify, 41, 42, 55, 59, 81
Lighting effects, 44, 44, 71-72, 74
Lighting registers, 44, 74, 78
Loadhi.sys, 84
Machine gun, 9, 9, 51, 53, 55
Machine gun ammo, 51, 53
Magnify icons, 10, 10
Make final, 76-77
Making your project, 11
Mask.vga, 83-83
MEM, 83, 94, 95
Memory, 7, 7-8, 16, 36, 38, 39, 48, 54, 61, 67, 69, 84, 86, 90-94, 96
Memory alloc fail, 91
Memory limitations, 61, 87
Message number, 30
midi, 49-50
Midi music files, 49
Misc, 18
Moving walls or whole rooms, 20
Music, 5, 86, 93
Musical note icon, 49, 49, 93
Numlist.vga, 83, 92
Object placing icons, 10
Object snap off, 19
Object won't show up on radar, 23
Open level, 75, 75
Outside scenarios, 10
P3DGCS directory, 6, 6, 12, 31, 36, 49, 52, 85
Palette, 12-12, 37, 41, 46, 49, 58, 66, 68, 71, 72, 83, 94
Paltable, 80-80
Pause, 83, 83
Pause.vga, 83
Pickups.txt, 40, 57-59
Pixels, 10-11, 36, 55-56, 59, 61, 64, 65, 67, 70, 82, 90
PKZIP, 79-80
Platforms, 10, 17, 26, 27, 43, 75
Play sound number, 43
Project directory, 11, 11, 12, 18, 44, 76, 80, 89, 92
pstr.exe, 31-33, 45, 89
pstr.txt, 30, 32, 33, 47, 53, 57, 89, 90, 93
Pygmy, 95, 95-96
QEMM, 84, 84
Quickness, 33, 33, 71
Quit, 66
Radar pack, 51
Radius, 43
RAM, 8, 16-17, 61, 67, 84, 86, 92, 96
Readouts, 83-83
Readouts.txt, 83, 83
Remove this animation, 43
Resistance to damage, 35
Resistance to forced entry, 24
Resolution, 11, 26, 60, 62, 70
Rocket ammo, 51
Rocket launcher, 37, 51
Rotating groups, 10, 21
Runcodes, 54-57
Save/load, 37
Sdone, 31, 32
Select and view icons, 9
Selecting objects, 19
Sentry, then hunt, 34
Set current level number, 75
Set operation, 42
Shading effects, 80
Shaped objects, 14
Shaped walls, 59
Shotgun, 9, 9
Skip if player close, 31, 44-45
Skip next if reg=val, 44
Sky, 5, 17-18, 28, 78
Slide, 25-26
SMARTDRV.SYS, 85
Solid color horizontal panels, 14
Solid walls, 14, 23, 63
Sound Blaster, 8, 49, 77
Sound number, 43
Sounds.txt, 44, 47
Special purpose registers, 27, 27
String, 31-33, 57, 93, 96
Symmetrical objects, 18, 18, 21, 24, 31, 41, 45-46, 50, 55, 64, 67, 74, 81, 93
Teleporters, 15
Test level, 78

Theaters.txt, 78, 95
Tree icon, 9, 10, 14, 17, 28, 31, 45
Trees, 10, 11, 13-14, 24, 28, 33, 52, 55, 61, 82
Trouble shooting, 7, 7, 88
Unconditional branch, 31, 31, 42, 43, 45
Universe registers, 15, 15, 27, 40, 43-44, 54, 58, 71
untitled, 7, 7
Vertical size, 28, 55, 62, 64, 96
VGA vs. VGR image files, 65
View, 8-10, 13, 14, 29, 34, 36-37, 68
Wall icons, 10, 28, 29
Warnings, 6
Warp-to point, 78, 78
Weapdef.txt, 40, 55, 58
Weapons, 9-9, 34, 37, 39-40, 50, 55, 58-59, 65, 78
Weapons directory, 38, 40, 55, 58, 81, 93
Weapons.txt, 38-40
What was the question?, 34
Windowed walls, 81
World sizes, 39, 56, 62
XMS, 39, 48, 83, 85, 92-94, 96
Zoom, 8, 14, 21, 41, 71, 75

pie in the sky software

Developers of 3D Entertainment Software

1596 Ayrault Rd. Fairport New York 14450

Phone 716-425-8782 Fax 716-425-8842

BBS 716-425-2962 Shipping 716-425-8510

INSTALLING THE GCS UNDER *WINDOWS 95*

Please follow these steps if you are using WINDOWS 95...

1. Click on the MSDOS prompt icon.

2. Put your PIE 3D GCS Install Disk #1 into drive A:

3. Type... A:INSTALL ...at the DOS prompt.

4. When the GCS installation is complete, type GCS to run the Game Creation System. If the GCS will not run, type HELP_ME for more information. (Please note that the underscore character "_" is typed by holding down the SHIFT key and pressing the "-" minus sign key, which is located next to the "=" equals sign key on your keyboard.)

pie in the sky software

GAME CREATION SYSTEM

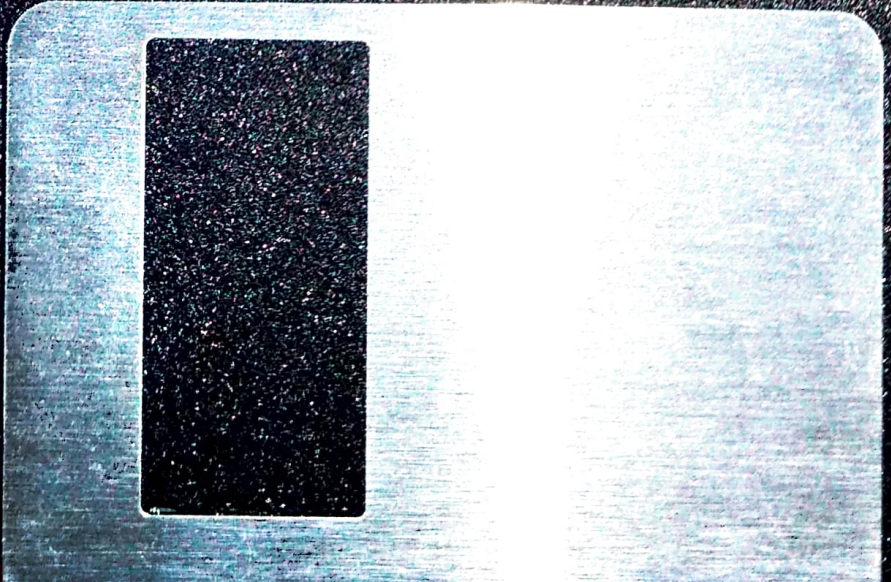
Disk #1 – GCS Install

pie in the sky software

GAME CREATION SYSTEM

Disk #2 – GCS Install

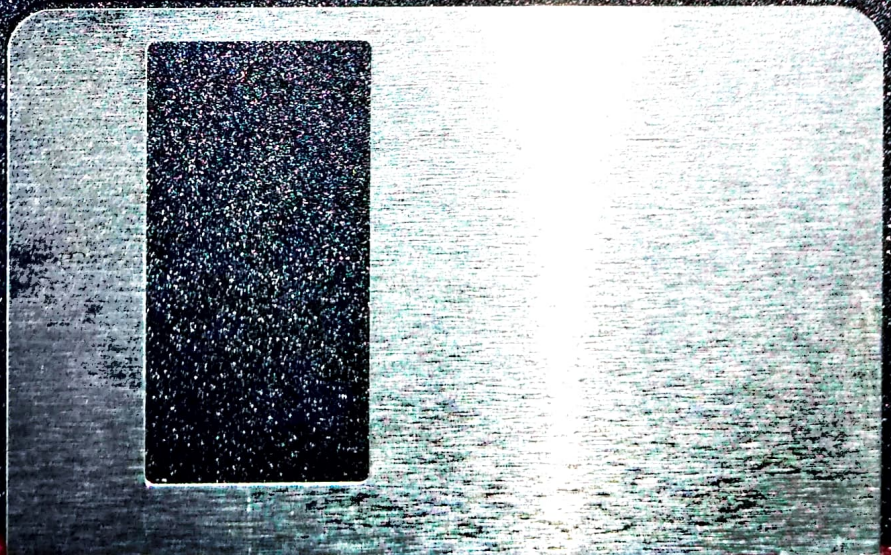
Disk #2 – Demo Install



pie in the sky software

GAME CREATION SYSTEM

Disk #3 – Demo Install
"Industrial Killers"



pie in the sky software

Developers of 3D Entertainment Software

Dear 3D Game Enthusiast!,

Pie in the Sky Software is delighted that you have expressed interest in our new line of 3D entertainment products. We can't wait to tell you about the Pie 3D Game Creation System and all it has to offer. This development kit is for a serious game player who has what it takes to become a *3D game designer*!

3D GAME CREATION SYSTEM

Pie in the Sky's 3D Game Creation System, the GCS, enables you to design and build your very own software entertainment products. This is no rinky-dink maze builder toy. This is an integrated program crammed with powerful features that help you build multi-level games, complete with weapons, sounds, music, and, of course, stunning 256 color, smooth-scrolling 3D action.

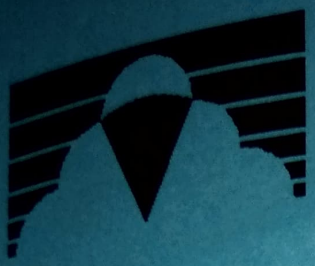
The GCS Editor is easy to use. Use your mouse to drag walls, objects, and enemy characters into position on a top-view grid. Test your layout with a simple mouse click. Our 3D game engine starts up and you are plunged into your own virtual reality! Bash open doors, race down hallways, hack through jungles,--all in first-person perspective. Run forward or backwards, spin, turn, jump, and leap in combat or chase situations.

Remember, you are the designer. Conjure a dark and sinister medieval castle. Engineer a sci-fi space station. Create a children's game. Do it all with the GCS's full-featured image workshop. Create new artwork or prepare existing images with GCSPAINT. You can even import images captured by your scanner! Imagine the fun of seeing yourself in a 3D world.

Keep track of all your images with the GCS's Object Library. Stockpile an impressive collection of objects, walls, and characters so you can use them in future games. Your work stays organized and at your fingertips.

FEATURES OF THE GCS

- solid walls of different sizes, placed at 90 or 45 degree angles
- walls with windows that you can see through
- shaped walls with uneven edges
- objects like trees, barrels, telephone poles, and more
- sliding doors that operate with keys
- textured floors and ceilings
- raised platforms
- interactive animated objects
- inventory items: health packs, ammo, armor, and more. Pick up, carry, use, and put down these items
- enemy characters that stand sentry, patrol, hunt, and attack. Adjust their strength and intelligence
- weapons: ninja kick, shotgun, machine gun, grenades, goo gun, and rocket launcher
- operational compass
- radar mapping
- damage/health indicator
- game play control panel instruments
- layout editing tools: select, identify, zoom, flip, rotate, copy, move, erase, grid snap, & raise/lower
- integrated paint program with special effects, palette-matching. Imports .PCX, .BMP, .GIF images
- fading, shading, and lighting effects
- text messages for players
- travel back and forth between levels
- pause and help screens
- 98 page manual with an index and a trouble-shooting section
- Diagnostic and debugging codes



pie in the sky software

ORDER FORM

These products are not available in stores. Mail this form to us so we can fill your order directly.

Name: _____

Company: _____

Address: _____

_____

Phone: _____

Place an X next to the products that you wish to order...

$6.00 - GCS DEMO "Industrial Killers"
$69.95 - Pie 3D Game Creation System
$34.95 - GCS MENU
$49.95 - GCS MEGA ART PACK
$34.95 - GCS MUSIC MIDI PACK
$24.95 - GCS INDUSTRIAL THEME ART PACK
$300.00 - GCS FOR C PROGRAMMERS
($230.05 plus $69.95 for regular GCS)

SHIPPING

It generally takes business 3 days to process your order. Place an X next to the shipping method that you prefer...

$4.00 shipping & handling US Priority Mail
$8.00 International Air Mail/Par Avion
$9.00 shipping & handling COD
COD not available for International
$16.00 shipping & handling Overnight Express Mail
$22.00 International Express Mail

PAYMENT

$ _____ Total cost. (NY residents add 8%)
Place an X next to your method of payment...

Check or money order payable to Pie in the Sky
C.O.D. (See fee above)
Credit card. Please fill in the information below...

MasterCard/VISA/AMEX #

_____

Exp. Date: _____/_____

Signature: _____

Thanks for your order!

SYSTEM REQUIREMENTS

Here are some technical details that you should consider. First, this is a DOS product. You can't use the GCS while running Windows. This program does not run on a Mac; it is strictly for a PC compatible. It will run on a 386 machine, but a 486 is recommended. You need 4 megs of RAM and at least 600K of DOS RAM free. This product comes on 3.5" floppy disks and it installs on your hard drive. A VGA graphics card is required and a SoundBlaster compatible sound card is necessary if you want music and sound effects. You'll also need a mouse. If your system meets these criteria, then you can be up and running the GCS in minutes!

SELL YOUR GAMES & KEEP THE PROFITS!

Any game that you craft with the GCS is totally yours. Put your finished games on floppies to share with friends. Upload your games to bulletin boards. Sell your games as shareware or strike a retail deal and sell your products on store shelves. That's right, no strings, royalties, licenses, or fees.

So, if you are ready to make the leap from 3D game player to 3D game designer, this is the product you've been waiting for!

OUR PRODUCTS

GCS DEMO "Industrial Killers" - This is a sample game that we created using several of the products discussed below. It has music & sound effects, title screens, a menu, mouse/joystick control, and multiple levels. It is designed to show off all the features of the GCS and GCS Add-on products. **$6.00**

3D GAME CREATION SYSTEM - This is a development kit for 3D game designers. It comes with a 98 page manual that guides you through the GCS's powerful capabilities. Layout your levels with our 3D world editor that works on a top-view grid. Click a button to use

GCSPAINT, a complete graphics workshop especially designed to create and prepare artwork for GCS-made games. Manage your project with our built-in Object Library. The heart of the GCS is Pie in the Sky's fast, smooth-scrolling, first-person perspective, 3D game engine. Distribute or market your GCS-made games and keep 100% of the profits. **$69.95**

GCS MENU - This GCS Add-on product gives your GCS-made game a professional look and feel.. Seamlessly add title screens, opening music, still full screen pictures, and animation sequences. The menu gives the player many options: start game, pause game, resume game, load game, save game, sound on/off, music on/off, mouse, joystick, and exit. **$34.95**

GCS MEGA ART PACK - This GCS Add-on product adds 6 megabytes of new artwork to your Object Libraries. You get walls, floors, ceilings, foliage, furniture, hostile human and non-human enemies, friendly characters, and dozens and dozens of additional pieces of artwork that are ideally suited for 3D games. Each image is already formatted, palette-matched, sized, and ready to put into your game with a simple mouse click! **$49.95**

GCS MUSIC MIDI PACK - Custom music helps you create the perfect mood for your games. This GCS Add-on product contains seven original musical compositions. Each tune was written by a professional musician especially for the GCS. Now each level of your game can have a distinctive atmosphere. **$34.95**

INDUSTRIAL THEME ART PACK - This is a GCS Add-on pack of artwork built around a high-tech industrial theme. You get walls, floors, ceilings, and objects that create a sinister, futuristic look. PLEASE NOTE: This art pack uses its own unique palette which is different from the palette of the GCS Basic Library and the Mega Pack Libraries. This artwork is not included in the Mega Pack. **$24.95**

All the products described above have a 30 day money back guarantee. Send us back the disks, manual, and box and we'll refund

your credit card account or send you a check for the cost of the product minus shipping.

GCS for C PROGRAMMERS - This product is designed for C programmers. You get all the features that come with the GCS plus you get specialized software tools, a library of functions and routines, as well as selected source code. An 81 page manual explains the nuts and bolts of the Pie 3D game engine. For example, the documentation covers the main animation loop, the zbuffer system, the FORTH AI system, the VERTS system, the MEMCHNK system, the explosion FIFO buffer, VGA page switching, critical errors, and more. C programmers can resize the view window, change the perspective to include looking up and down, set point lighting, customize the inventory system, the weapons, the control panel, and the motion of the player. This product is perfect for programmers who want to develop original projects without getting bogged down in all the drudgery of building a 3D game engine from scratch. **$300.00** ($230.05 w/ $69.95 GCS)

MORE INFORMATION -

For more information about our company and our products, we invite you to check out our World Wide Web page at...

http://www.psky.com

Pie in the Sky software gives independent 3D game designers a chance to break into an exploding field that has, up until now, been dominated by a few big companies. Wouldn't you rather spend your time creating something new and original rather than spending your time hanging out in computer superstores waiting for the next 3D title to be released? It's time to put all your expertise as a veteran game player to good use! With our tools, you can build the games you've always dreamed of playing. Pack your projects with action, suspense, strategy, and fun. We encourage you to take advantage of this opportunity. We hope to receive your order soon!

Pie in the Sky Software
1596 Ayrault Rd.
Fairport NY 14450
1-800-537-3344